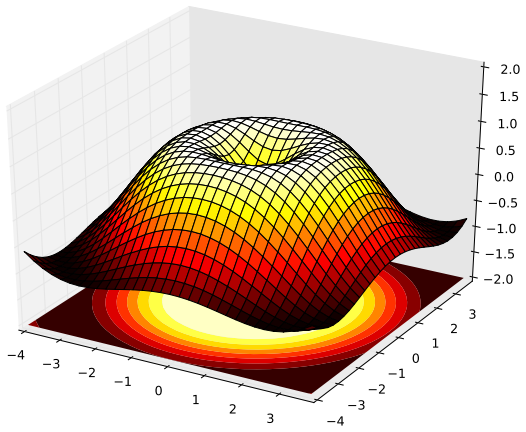
Informatique

Saint Joseph La Salle LORIENT

Marc Beutier
obelix56@free.fr
http://obelix56.free.fr

cpge TSI

Établissement St Joseph - LaSalle Lorient



book_20-06_python_eleve 30 juin 2020

Python & Scilab

Version Élève

Ce fascicule s'adresse avant tout à vous, étudiants de première année de CPGE. J'espère qu'il pourra vous servir de base pour l'année et pourquoi pas pour les années suivantes ?
Il sera mis à jour sur

http://obelix56.free.fr

Vous trouverez également sur ce site les dossiers *eleve* dans lequels se trouvent tous les programmes écrits ici en *python* (*.py*) et *scilab* (*.sci* et *.sce*).
Les programmes proposés durant les activités et ceux demandés en exercices ne sont pas disponibles, pour le bon déroulement des séances ...
La progression de l'année sera calquée sur ce fascicule et comme nous ne trouvons le temps de tout faire, vous pourrez combler vos lacunes le soir, avant de vous endormir ...

Le fascicule existe rassemble le programme des deux années en algorithmique, sur *Scilab* , sur *Python* et sur le traitement d'images.

Je ne peux remercier toutes les personnes qui m'ont aidé à rédiger ce fascicule. Entre les collègues, les auteurs d'ouvrages de référence, les tutoriels, FAQ et autres forums, je tiens particulièrement à remercier Arnaud Coatanhay et Christophe Osswald, enseignants - chercheurs à l'ENSTA Bretagne, pour leur disponibilité et la qualité de leur formation en *Python* et Philippe Roux, Jean-Paul Chehab, Jérôme Briot et Michael Baudin pour le partage de leurs ouvrages en format LaTeXconcernant *Scilab*.
"Une introduction à Python 3" écrit par Bob Cordeau et Laurent Pointal a aussi certainement inspiré mon travail. Leurs sources LaTeXet *Python* sont disponibles sur leur site. On peut trouver cela ici :

http://perso.limsi.fr/pointal/python:courspython3

Pour ce qui est des forums, celui que j'utilise régulièrement est :

http://www.developpez.net/forums/f96/autres-langages/python-zope/

En cas de réclamation, veuillez me contacter à l'adresse indiquée sur la première page.

Table des matières

Partie 1

Algorithmique

Table des matières

1

Introduction

1.1 Compétences exigées

Objectifs

Les capacités évaluées dans cette partie de la formation sont :

- comprendre un algorithme et expliquer ce qu'il fait,
- modifier un algorithme existant pour obtenir un résultat différent,
- concevoir un algorithme répondant à un problème précisément posé,
- expliquer le fonctionnement d'un algorithme,
- écrire des instructions.

1.2 Qu'est-ce qu'un algorithme ?

Algorithme

Un algorithme est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.

1.3 Les algorithmes dans la vie courante

Les manuels d'utilisation des appareils actuels de la vie courante sont essentiellement des recueils d'algorithmes : des instructions sont données afin de faire fonctionner une fonction quelconque.
Les exemples de la vie courante ne manquent pas : automobiles, appareils divers, ...

Une recette de cuisine est également un algorithme simple.
Celui-ci comporte grossièrement trois étapes :

1. Réunir les ingrédients
2. Préparer
3. Déguster

La préparation consiste à exécuter une suite d'instructions : par exemple, plonger les tomates dans une casserole d'eau bouillante pendant quelques instants avant de les peler. On ne sait pas pourquoi il faut procéder de la sorte et d'ailleurs, ça n'a aucune importance : la recette a été écrite par quelqu'un qui sait. Elle marche.

En comparant avec les algorithmes de mathématiques, on pourrait dire que les ingrédients de la recette sont les **entrées** du processus auxquelles on applique le **traitement** (la préparation) pour obtenir, en **sortie**, un plat que l'on dégustera.

1.4 Construction d'un algorithme

En langage naturel, un algorithme se présente en général sous la forme suivante :

- Déclaration des variables :
  On décrit dans le détail les éléments que l'on va utiliser dans l'algorithme.
- Initialisation et / ou Entrée des données :
  On récupère les données et/ou on les initialise.
- Traitement des données :
  On effectue les opérations nécessaires pour répondre au problème posé.
- Sortie :
  On affiche le résultat.

On pourra alors mettre l'algorithme sous la forme suivante :

Algorithme : construction

```
1  VARIABLES
2  Les variables (entrées et sorties entre autres) ainsi que leur type
3  ENTRÉES
4  LIRE les entrées
5  SORTIES
6  AFFICHER les sorties
7  DEBUT_ALGORITHME
8  FIN_ALGORITHME
```

Vous pourrez, pour vous entraîner, télécharger le logiciel libre *AlgoBox* à l'adresse suivante :

http://www.xm1math.net/algobox/download.html

Il est disponible pour tous les sytsèmes d'exploitation.

1.5 Différents langages

Il existe une quantité de langages de programmation et de logiciels permettant de définir des algorithmes. Cette année, nous serons amenés à utiliser les outils suivants :

- Langage de programmation **Python**
- Logiciel **Scilab**

Considérons un algorithme historiquement célèbre, l'algorithme d'Euclide, qui sert à calculer le plus grand commun diviseur (pgcd) :
Étant donnés deux entiers, retrancher le plus petit au plus grand et recommencer jusqu'à ce que les deux nombres soient égaux. La valeur obtenue est le plus grand diviseur commun. L'idée de départ est de prendre les 2 nombres et tant qu'ils ne sont pas égaux, de retirer le plus petit au plus grand.
L'algorithme se présente sous la forme suivante :

Algorithme d'Euclide

```
1   VARIABLES
2   a : int # "int" signifie entier comme integer
3   b : int
4   DEBUT_ALGORITHME
5      TANT_QUE a et b sont différents FAIRE
6         DEBUT_TANT_QUE
7         SI a est le plus grand ALORS
8            DEBUT_SI
9            remplacer a par a - b
10             FIN_SI
11          SI b est le plus grand ALORS
12             DEBUT_SI
13             remplacer b par b - a
14             FIN_SI
15          FIN_TANT_QUE
16       AFFICHER "le pgcd est" a
17    FIN_ALGORITHME
```

Voici cet algorithme présenté dans différents langages de programmation :

```
# CODE PYTHON
def pgcd(a,b):
        while a!=b # ou while a
           <>b:
                if a>b : a=a-b
                else : b=b-a
        return a
```

```
# CODE JAVA
public static int pgcd(int a,
    int b) {
        while (a!=b){
                if (a>b) a=a-b
                else b=b-a
        }
        return a;
}
```

```
(* Code OCaml *)
let rec pgcd(a, b) =
        if a = b then a else
                if a > b then
                   pgcd(a-b, b)
                else pgcd(a,b-a)
        ;;
```

```
Rem Code OOBasic
Function Pgdc(ByVal a As Integer
    ,
                        ByVal b
                           As
                           Integ r
                           ) As

                           Integ r


        Do While a<>b
                If a>b Then
                        a=a-b
                Else
                        b=b-a
                EndIf
        Loop
        Pgdc=a
End Function
```

```
# CODE OCTAVE
function r = pgcd(a,b)
        while (a ~=b)
                if (a>b) : a=a-b
                   ;
                else : b=b-a;
        end
        r=a;
```

```
/* CODE C */
int pgdc(int a,int b)
{
while (a != b) {
if (a>b) a=a-b;
else b=b-a; }
return a;
}
```

```
# CODE RUBY
def pgdc(a,b)
        while a!=b
                if a>b then a=a-
                   b
                else b=b-a end
        end
        a
end
```

```
; Code Scheme
(define (pgdc a b)
        (cond
                ((< a b) (pgdc a
                   (- b a) ))
                ((> a b) (pgdc
                   (- a b) b ))
                (else a)
        )
)
```

```
# CODE PERL

sub pgcd {
    my ($a, $b) = @_;
    while ($a != $b) {
        if ($a > $b) {
            $a = $a - $b;
        } else {
            $b = $b - $a;
        }
    }
    return $a;
}
```

2

Action !

2.1 Un premier exemple complet

Le but de ce premier algorithme consiste à déterminer la distance entre deux points connaissant leurs coordonnées dans un repère orthonormé.

On considère les points $A\begin{pmatrix}x_A\\y_A\end{pmatrix}$ et $B\begin{pmatrix}x_B\\y_B\end{pmatrix}$ définis dans un repère orthonormal $(O,\vec{\imath},\vec{\jmath})$.
Construire un algorithme permettant de calculer la longueur AB.

2.1.1 Langage naturel

Le langage naturel, pour nous, est le français. Nous utiliserons seulement des mots simples, le texte doit être clair et bien structuré.
On sait que la longueur d'un segment AB est définie par : $AB=\sqrt{(x_B-x_A)^2+(y_B-y_A)^2}$.

On peut construire l'algorithme suivant :

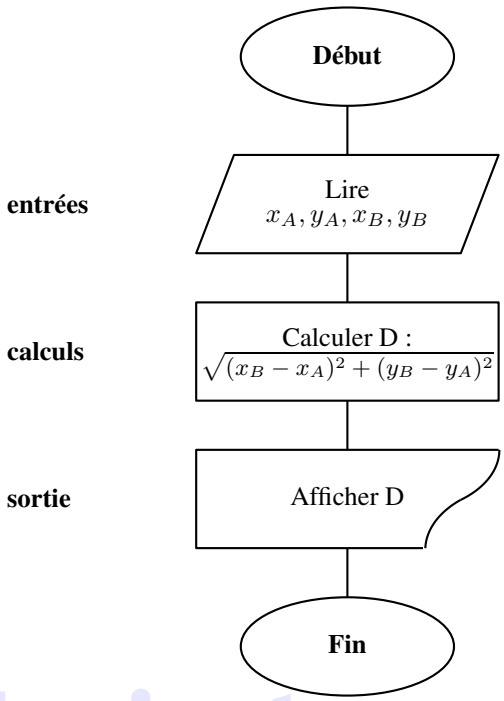
Variables :
x_A est l'abscisse de A
y_A est l'ordonnée de A
x_B est l'abscisse de B
y_B est l'ordonnée de B
D est la distance entre A et B

Initialisation, entrées :
Saisir x_A
Saisir y_A
Saisir x_B
Saisir y_B

Traitement :
D prend la valeur
$\sqrt{(x_B-x_A)^2+(y_B-y_A)^2}$

Sortie :
Afficher la valeur de D

Ou sous forme d'organigramme :



2.1.2 Avec Python

Python est un **langage de programmation** facile à utiliser et puissant. Il offre des structures de données de haut niveau et une approche simple mais réelle de la programmation.
Il est téléchargeable à l'adresse :

http://www.python.org/download/

L'algorithme précédent peut s'écrire en python de la façon suivante :

distance1.py

```
1  # -*- coding: utf-8 -*-
2
3  from math import sqrt
4  # ou from math import *
5
6  # commentaire : entrée des données
7  print ("Entrez l'abscisse de A")
8  x_A = float(input())
9  print ("Entrez l'ordonnée de A")
10 y_A = float(input())
11 print ("Entrez l'abscisse de B")
12 x_B = float(input())
13 print ("Entrez l'ordonnée de B")
14 y_B = float(input())
15 # calcul de la distance (commentaires)
16 d = sqrt((x_B-x_A)**2+(y_B-y_A)**2)
17 # affichage du résultat
18 print ("La distance entre A et B est :")
19 print (d)
20 # affichage du résultat arrondi
21 print("Avec 2 décimales, cela donne : %.2f" %d)
```

2.2 À vous de jouer !

2.2.1 Milieu de 2 points

⇒ **Activité 2. 1**
Construisez un algorithme en langage naturel qui permet de déterminer les coordonnées du milieu I d'un segment $[AB]$ connaissant les coordonnées de A et de B dans un repère quelconque.

2.2.2 Résolution mathématique

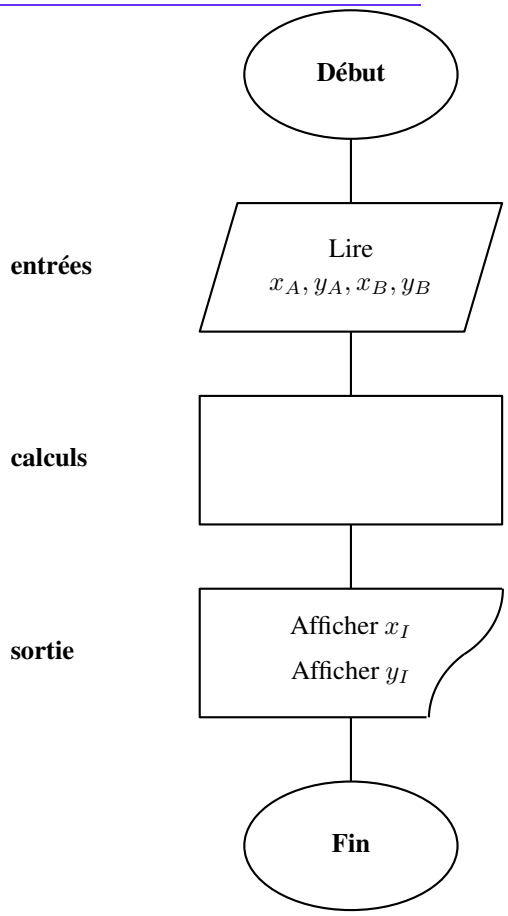
2.2.3 En langage naturel

Variables :

x_A est l'abscisse de A
y_A est l'ordonnée de A
x_B est l'abscisse de B
y_B est l'ordonnée de B
x_I est l'abscisse du milieu I du segment $[AB]$
y_I est l'ordonnée du milieu I du segment $[AB]$

Initialisation, entrées :

Saisir x_A
Saisir y_A
Saisir x_B
Saisir y_B



2.2.4 Avec Python, si vous avez déjà quelques notions

milieu1.py

```
1  # -*- coding: utf-8 -*-
2
3  from math import *
4
5  # commentaire : entrée des données
6  print ("Entrez l'abscisse de A")
7  x_A = float(input())
8  print ("Entrez l'ordonnée de A")
9  y_A = float(input())
10 print ("Entrez l'abscisse de B")
11 x_B = float(input())
12 print ("Entrez l'ordonnée de B")
13 y_B = float(input())
14
15 # calcul des coordonnées du milieu
16 x_I = (x_A+x_B)/2
17 y_I = (y_A+y_B)/2
18
19 # affichage du résultat
20 print ("Le milieu a pour coordonnées :")
21 print (x_I, y_I)
22 # affichage du résultat arrondi
23 print("Avec 2 décimales, cela donne \
24 %.2f pour l'abscisse et \
```

```
25  %.2f pour l'ordonnée" %(x_I,y_I))
26  # \ permet de passer à la ligne
```

2.3 Algorithmique : techniques de base

2.3.1 Variables et affectations

Les variables en algorithmique

- Les variables algorithmiques peuvent servir à stocker des données de différents types.
- La valeur d'une variable peut changer au fil des instructions de l'algorithme.
- Les opérations sur les variables s'effectuent ligne après ligne et les unes après les autres.
- Quand l'ordinateur exécute une ligne du type `mavariable PREND_LA_VALEUR un calcul`, il effectue d'abord le calcul et stocke ensuite le résultat dans `mavariable`.

Remarque

Les commentaires seront placés après le symbole # comme dans python pour faciliter la lecture.

⇒ **Activité 2. 2**

On considère l'algorithme suivant :

Valeurs de x, y z

```
1  VARIABLES
2  x : int # "int" signifie entier comme integer
3  y : int
4  z : int
5  DEBUT_ALGORITHME
6     x ← 2 # x prend la valeur 2
7     y ← 3
8     z ← x + y
9  FIN_ALGORITHME
```

Après exécution de l'algorithme, la variable x contient la valeur ☐, la variable y contient la valeur ☐ et la variable z contient la valeur ☐.

⇒ **Activité 2. 3**

On considère l'algorithme suivant :

Valeur de x

```
1  VARIABLES
2  x : int
3  DEBUT_ALGORITHME
4     x ←2
5     x ← x + 1
6  FIN_ALGORITHME
```

Après exécution de l'algorithme, la variable x contient la valeur : ☐.

⇒ **Activité 2.4**

Ajoutons la ligne « $x \leftarrow 4 * x$ » à la fin du code précédent. x contient alors la valeur ☐.

⇒ **Activité 2.5**

On considère l'algorithme suivant :

Valeur de y

```
1   VARIABLES
2   y : int
3   DEBUT_ALGORITHME
4      y ← 2
5      y ← y + 1
6      y ← 4 * y
7   FIN_ALGORITHME
```

Après exécution de l'algorithme, la variable y contient la valeur : ☐.

⇒ **Activité 2.6**

On considère l'algorithme suivant :

Valeurs de a, b, c

```
1   VARIABLES
2   a : int
3   b : int
4   c : int
5   DEBUT_ALGORITHME
6      a ← 5
7      b ← 3
8      c ← a + b
9      b ← b + a
10     a ← c
11  FIN_ALGORITHME
```

Après exécution de l'algorithme, la variable a contient la valeur ☐, la variable b contient la valeur ☐ et la variable c contient la valeur ☐.

⇒ **Activité 2.7**

On considère l'algorithme suivant :

Afficher z

```
1   VARIABLES
2   x : int
3   y : int
4   z : int
5   ENTRÉES
6   LIRE x
7   SORTIES
8   AFFICHER z
9   DEBUT_ALGORITHME
10     y ← x − 2
11     z ← −3 * y − 4
12     AFFICHER z
13  FIN_ALGORITHME
```

On cherche maintenant à obtenir un algorithme équivalent sans utiliser la variable y. Complétez la ligne 9 dans l'algorithme ci-dessous pour qu'il réponde au problème.

Afficher z simplifié

```
1   VARIABLES
2   x : int
3   z : int
4   ENTRÉES
5   LIRE x
6   SORTIES
7   AFFICHER z
8   DEBUT_ALGORITHME
9      z ← .......
10      AFFICHER z
11   FIN_ALGORITHME
```

2.3.2 Instructions conditionnelles

SI...ALORS...SINON

Comme nous l'avons vu ci-dessus, un algorithme permet d'exécuter une liste d'instructions les unes à la suite des autres. Mais on peut aussi "dire" à un algorithme de n'exécuter des instructions que si une certaine condition est remplie. Cela se fait grâce à la commande `SI...ALORS` :

```
SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
```

Il est aussi possible d'indiquer en plus à l'algorithme de traiter le cas où la condition n'est pas vérifiée avec la commande `SINON`. On obtient alors la structure suivante :

```
SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
SINON
  DEBUT_SINON
  ...
  FIN_SINON
```

⇒ **Activité 2.8**

On cherche à créer un algorithme qui demande un nombre à l'utilisateur et qui affiche la racine carrée de ce nombre s'il est positif. Complétez la ligne 9 dans l'algorithme ci-dessous pour qu'il réponde au problème.

Racine carrée

```
1   VARIABLES
2   x : int
3   racine : float
4   ENTRÉES
5   LIRE x
6   SORTIES
7   AFFICHER racine
8   DEBUT_ALGORITHME
9      SI (.........) ALORS
10        DEBUT_SI
11        racine ← sqrt(x)
12        AFFICHER racine
13        FIN_SI
14   FIN_ALGORITHME
```

⇒ **Activité 2.9**

On cherche à créer un algorithme qui demande à l'utilisateur d'entrer deux nombres entiers (stockés dans les variables x et y) et qui affiche le plus grand des deux. Complétez les lignes 12 et 16 dans l'algorithme ci-dessous pour qu'il réponde au problème.

Comparaison

```
1   VARIABLES
2   x : int
3   y : int
4   ENTRÉES
5   LIRE x
6   LIRE y
7   SORTIES
8   AFFICHER le plus grand des deux (x, y)
9   DEBUT_ALGORITHME
10      SI (x > y) ALORS
11         DEBUT_SI
12         AFFICHER .......
13         FIN_SI
14      SINON
15         DEBUT_SINON
16         AFFICHER ......
17         FIN_SINON
18  FIN_ALGORITHME
```

⇒ **Activité 2. 10**

On considère l'algorithme suivant :

Valeurs de a et b

```
1   VARIABLES
2   a : int
3   b : int
4   DEBUT_ALGORITHME
5      a ← 1
6      b ← 3
7      SI (a > 0) ALORS
8         DEBUT_SI
9         a ← a + 1
10        FIN_SI
11     SI (b > 4) ALORS
12        DEBUT_SI
13        b ← b − 1
14        FIN_SI
15  FIN_ALGORITHME
```

Après exécution de l'algorithme,

- la variable a contient la valeur : ☐,
- La variable b contient la valeur : ☐.

⇒ **Activité 2. 11**

On cherche à concevoir un algorithme correspondant au problème suivant :

- on demande à l'utilisateur d'entrer un nombre (représenté par la variable x)
- si le nombre entré est différent de 1, l'algorithme doit stocker dans une variable y la valeur de $\dfrac{1}{x-1}$ et afficher la valeur de y
  (note : la condition x différent de 1 s'exprime avec le code $x! = 1$). On ne demande pas de traiter le cas contraire.

Complétez l'algorithme ci-dessous pour qu'il réponde au problème.

Inverse

```
1   VARIABLES
2   x : int
3   y : int
4   ENTRÉES
5   LIRE .....
6   SORTIES
7   AFFICHER ......
8   DEBUT_ALGORITHME
9      SI (.........) ALORS
10        DEBUT_SI
11        ..... ← .....
12        AFFICHER .....
13        FIN_SI
14   FIN_ALGORITHME
```


2.3.3 Boucles

Boucles POUR...DE...A

- Les boucles permettent de répéter des instructions autant de fois que l'on souhaite.
- Lorsqu'on connaît par avance le nombre de fois que l'on veut répéter les instructions, on utilise une boucle du type POUR...DE...A... dont la structure est la suivante :

```
POUR...ALLANT_DE...A...
  DEBUT_POUR
  ...
  FIN_POUR
```

- Exemple : l'algorithme ci-dessous permet d'afficher la racine carrée de tous les entiers de 1 jusqu'à 50. La variable n est appelée « compteur de la boucle ».

Racine carrée 2

```
1   VARIABLES
2   n : int
3   racine : float
4   SORTIES
5   AFFICHER racine
6   DEBUT_ALGORITHME
7      POUR n ALLANT_DE 1 A 50
8         DEBUT_POUR
9         racine ← sqrt(n) # ou racine ← √n
10          AFFICHER racine
11          FIN_POUR
12   FIN_ALGORITHME
```

Remarques

- La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
- Sauf précision, cette variable est automatiquement augmentée de 1 à chaque fois.
- On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais les instructions comprises entre `DEBUT_POUR` et `FIN_POUR` ne doivent pas modifier la valeur de la variable qui sert de compteur.

⇒ **Activité 2. 12**

On cherche à concevoir un algorithme qui affiche, grâce à une boucle `POUR...DE...A`, les résultats des calculs suivants : $8*1$, $8*2$, $8*3$, $8*4$, ... jusqu'à $8*10$.
La variable n sert de compteur à la boucle et la variable *produit* sert à stocker et afficher les résultats. Complétez les lignes 7 et 9 dans l'algorithme ci-dessous pour qu'il réponde au problème :

Table de 8

```
1   VARIABLES
2   n : int
3   produit : int
4   SORTIES
5   AFFICHER produit
6   DEBUT_ALGORITHME
7      POUR n ALLANT_DE .... A .....
8         DEBUT_POUR
9         produit ← .......
10          AFFICHER produit
11          FIN_POUR
12   FIN_ALGORITHME
```

⇒ **Activité 2. 13**

On considère l'algorithme suivant :

Somme 100

```
1   VARIABLES
2   n : int
3   somme : int
4   SORTIES
5   AFFICHER somme
6   INITIALISATION
7   somme ← 0
8   DEBUT_ALGORITHME
9      POUR n ALLANT_DE 1 A 100
10        DEBUT_POUR
11        somme ← somme + n
12        FIN_POUR
13     AFFICHER somme
14  FIN_ALGORITHME
```

Complétez les phrases suivantes :

- Après exécution de la ligne 7, la variable `somme` contient la valeur : ☐.
- Lorsque le compteur n de la boucle vaut 1 et après exécution du calcul de la ligne 11, la variable *somme* vaut : ☐.
- Lorsque le compteur n de la boucle vaut 2 et après exécution du calcul de la ligne 11, la variable *somme* vaut : ☐.
- Lorsque le compteur n de la boucle vaut 3 et après exécution du calcul de la ligne 11, la variable *somme* vaut : ☐.

Que permet de calculer cet algorithme ?
☐.

⇒ Activité 2. 14

Complétez les lignes 9 et 11 de l'algorithme ci-dessous pour qu'il permette de calculer la somme $5^2 + 6^2 + 7^2 + \cdots + 24^2 + 25^2$, c'est-à-dire la somme $\sum_5^{25} n^2$.

Somme des carrés

```
1   VARIABLES
2   n : int
3   somme : int
4   SORTIES
5   AFFICHER somme
6   INITIALISATION
7   somme ← 0
8   DEBUT_ALGORITHME
9      POUR n ALLANT_DE .... A ......
10         DEBUT_POUR
11         somme ← somme+......
12         FIN_POUR
13     AFFICHER somme
14  FIN_ALGORITHME
```

Boucles TANT QUE...

- Il n'est pas toujours possible de connaître par avance le nombre de répétitions nécessaires à un calcul. Dans ce cas là, il est possible d'avoir recours à la structure `TANT QUE...` qui se présente de la façon suivante :

```
TANT_QUE...FAIRE
  DEBUT_TANT_QUE
  ...
  FIN_TANT_QUE
```

  Cette structure de boucle permet de répéter une série d'instructions (comprises entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE`) tant qu'*une certaine condition est vérifiée*.
- Exemple : Comment savoir ce qu'il reste si on enlève 25 autant de fois que l'on peut au nombre 583 ? Pour cela on utilise une variable n, qui contient 583 au début, à laquelle on enlève 25 tant que c'est possible, c'est à dire tant que n est supérieur ou égal à 25. Voici ci-dessous un exemple d'algorithme avec une structure de boucle.

Tant que

```
VARIABLES
n : int
SORTIES
AFFICHER n
INITIALISATION
n ← 583
DEBUT_ALGORITHME
   TANT_QUE n >= 25 FAIRE
      DEBUT_TANT_QUE
        n ← n − 25
        FIN_TANT_QUE
     AFFICHER n
FIN_ALGORITHME
```

Remarques

- Si la condition du `TANT QUE` est fausse dès le début, les instructions entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` ne sont jamais exécutées (la structure TANT QUE ne sert alors strictement à rien).
- Il est indispensable de s'assurer que la condition du `TANT QUE` finisse par être vérifiée (le code entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme ne pourra pas fonctionner.

⇒ Activité 2. 15

On cherche à connaître le plus petit entier N tel que 2^N soit supérieur ou égal à 10000. Pour résoudre ce problème de façon algorithmique :

- On utilise une variable N à laquelle on donne au début la valeur 1.
- On augmente de 1 la valeur de N tant que 2^N n'est pas supérieur ou égal à 10000.

Une structure `TANT QUE` est particulièrement adaptée à ce genre de problème car on ne sait pas a priori combien de calculs seront nécessaires.
Compléter les lignes 8 et 10 de l'algorithme ci-dessous pour qu'il réponde au problème :

2 puissance N

```
1   VARIABLES
2   N : int
3   SORTIES
4   AFFICHER N
5   INITIALISATION
6   N ← 1
7   DEBUT_ALGORITHME
8      TANT_QUE 2^N ..... FAIRE
9         DEBUT_TANT_QUE
10          N ← .....
11          FIN_TANT_QUE
12       AFFICHER N
13  FIN_ALGORITHME
```

⇒ Activité 2.16

On considère le problème suivant :

- On lance une balle d'une hauteur initiale de 3 m.
- On suppose qu'à chaque rebond, la balle perd 10% de sa hauteur (la hauteur est donc multipliée par 0,9 à chaque rebond).
- On cherche à savoir le nombre de rebonds nécessaire pour que la hauteur de la balle soit inférieure ou égale à 10 cm.

Complétez les lignes 10 et 13 de l'algorithme ci-dessous pour qu'il réponde au problème.

Nombre de rebonds

```
1   VARIABLES
2   nombre_rebonds : int
3   hauteur : float
4   SORTIES
5   AFFICHER nombre_rebonds
6   INITIALISATION
7   nombre_rebonds ← 0
8   DEBUT_ALGORITHME
9      hauteur ← 300
10     TANT_QUE (hauteur.....) FAIRE
11        DEBUT_TANT_QUE
12        nombre_rebonds ← nombre_rebonds + 1
13        hauteur ← ..........
14        FIN_TANT_QUE
15     AFFICHER nombre_rebonds
16  FIN_ALGORITHME
```

⇒ Activité 2.17

Écrivez un algorithme qui prend en entrée un entier naturel a et affiche cet entier écrit à l'envers. Par exemple, si $a = 1234$, la fonction devra retourner $a = 4321$. Vous pourrez utiliser les fonctions `quotient(`n`,` p`)` et `reste(`n`,` p`)` qui donnent le quotient et le reste de la division de n par p.

L'idée est que si l'on prend le reste de a dans la division par 10, on récupère le dernier chiffre, et si on prend le quotient, on récupère a privé de son dernier chiffre. Il suffit d'itérer le procédé, en décalant à chaque fois le résultat provisoire vers la gauche.

Renversement

```
1   VARIABLES
2
3
4   ENTRÉES
5   LIRE a
6   SORTIES
7   AFFICHER b "inverse" de a
8   INITIALISATION
9
10   DEBUT_ALGORITHME
11      TANT_QUE ..................... FAIRE
12         DEBUT_TANT_QUE
13
14
15
16         FIN_TANT_QUE
17      AFFICHER b
18   FIN_ALGORITHME
```

3

Exercices d'algorithmique

► Exercice 3. 1 : Premiers nombres entiers : algorithme

Écrivez un algorithme qui demande à l'utilisateur de saisir un nombre entier n puis affiche les n premiers nombres entiers.

► Exercice 3. 2 : Premiers entiers impairs : algorithme

Écrivez un algorithme qui demande à l'utilisateur de saisir un nombre entier n puis affiche les n premiers entiers impairs.

► Exercice 3. 3 : Somme des premiers entiers pairs : algorithme

Écrivez un algorithme qui demande à l'utilisateur de saisir un nombre entier n puis calcule la somme des n premiers entiers pairs en commençant par 2.

► Exercice 3. 4 : Intérêts : algorithme

Écrivez un algorithme qui, à partir d'un montant à épargner et un taux d'intérêt annuel, calcule le montant augmenté des intérêts pour les n années à venir.

► Exercice 3. 5 : Année bissextile : algorithme

Écrivez un algorithme qui détermine si une année n est bissextile.
Une année bissextile n compte 366 jours et une année non bissextile 365 jours. Une année n est bissextile si elle est divisible par 4 sauf si elle est séculaire, c'est-à-dire s'il s'agit d'un siècle.
Les siècles, années terminées par deux zéros, ne sont, en général, pas bissextiles sauf si elles sont divisibles par 400.
Quelques exemples : 1996 était bissextile, 1200 et 2000 également, 1997 ne l'était pas, 1800 et 1900 non plus mais 2400 le sera.

On pourra s'aider du diagramme de Venn suivant :

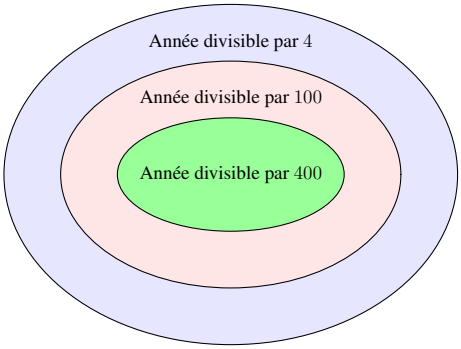


TABLE 3.1 – Diagramme de Venn : année bissextile

En réalité, il faudrait aussi étudier le cas des années divisibles par 4000 qui ne sont pas bissextiles mais d'ici là, le mouvement des planètes aura peut-être changé !

Vous pourrez vous aider pour vérifier l'algorithme d'un tableau comme ci-dessous :

	condition 1	condition 2	condition 3	année
2013				
2014				
2016				
2020				
2000				
1900				
2100				

▶ Exercice 3. 6 : Impôt sur les bénéfices : algorithme

Écrivez un algorithme qui calcule l'impôt sur le bénéfice d'une société, le montant du bénéfice étant demandé à l'utilisateur, le montant de l'impôt étant de 20 % si le bénéfice est inférieur à 10000 €, de 2000 + 25 % si le bénéfice est compris entre 10000 et 15000 € et de 3000 + 30 % si le bénéfice est supérieur à 15000 €.

▶ Exercice 3. 7 : Devinette 1 : algorithme

Écrivez un algorithme dans lequel l'utilisateur doit deviner un nombre pair compris entre 10 et 100 généré par l'ordinateur avec la fonction :
`nombreAléatoire(entier` n`) : entier`. Cette fonction génère un nombre compris entre 0 et n de façon aléatoire.
L'ordinateur pourra répondre aux propositions de l'utilisateur par "trop grand" ou "trop petit".

▶ Exercice 3. 8 : Devinette 2 : algorithme

Écrivez un algorithme dans lequel l'ordinateur devine un nombre pair entre 0 et 100 choisi par l'utilisateur (version dichotomique).
L'utilisateur pourra répondre par "+" si la proposition de l'ordinateur est trop petite, "-" si elle est trop grande et "=" si elle est exacte.

▶ Exercice 3. 9 : Échange de deux variables

Écrivez un algorithme qui échange le contenu de deux variables.

▶ Exercice 3. 10 : Recherche dans un tableau

Écrivez un algorithme cherchant dans un tableau le nombre 20.

4

Solutions d'algorithmique

Partie 2

Python année 1

Table des matières

5

Python en mode calculatrice

5.1 Compétences exigées

Objectifs

La capacité évaluée dans cette partie de la formation est :

- choisir un type de données en fonction d'un problème à résoudre.

5.2 Manipuler des nombres

En première approche, le langage *Python* peut être vu comme un ensemble de commandes simples à utiliser. En ce sens, il peut jouer le rôle d'une calculatrice scientifique classique. Pour s'en convaincre il suffit de lancer la séquence triviale suivante :

```
>>> a=1
>>> b=2
>>> c=a+b
```

Pour avoir le résultat de l'opération, on prolongera simplement par :

```
>>> print (c)
3
```

Remarque

La fonction `print()` sera vu prochainement (page 59) : elle permet d'afficher un résultat mais n'est pas indispensable en mode calculatrice.

Plus simplement, en console, on peut faire :

```
>>> c
3
```

5.3 Affectation

5.3.1 Affectation simple

On vient de le voir, l'affectation se fait avec le signe d'égalité = qui n'a rien à voir avec le signe d'égalité mathématique. En effet, les instructions :

```
x = 0
x = x + 1
```

n'ont aucune signification mathématique alors qu'en *Python*, on incrémente (on augmente) de 1 la variable x.

5.3.2 Affectations simultannées ou multiples

Il est possible d'effectuer des affectations simultanées :

```
>>> a,b,c = 5,9,12
>>> a
5
>>> b
9
>>> c
12
```

ou multiples :

```
>>> x = y = 2
>>> y = y + 2
>>> y
4
>>> x
2
```

⚠ N'utilisez pas l'affectation multiple pour les listes car celles-ci seraient alors stockées à la même adresse, ce qui aurait pour conséquence de changer les 2 listes lors de la modification d'une des deux : cf page 54.

5.4 Variables

L'essentiel du travail effectué par un programme d'ordinateur consiste à manipuler des données. Ces données peuvent être très diverses, mais dans la mémoire de l'ordinateur, elles se ramènent toujours en définitive à une suite finie de nombres binaires.

Pour pouvoir accéder aux données, le programme d'ordinateur (quel que soit le langage dans lequel il est écrit) fait abondamment usage d'un grand nombre de variables de différents types.

Une variable apparaît dans un langage de programmation sous un nom de variable à peu près quelconque, mais pour l'ordinateur il s'agit d'une référence désignant une adresse mémoire, c'est-à-dire un emplacement précis dans la mémoire vive.

À cet emplacement est stockée une valeur bien déterminée. C'est la donnée proprement dite, qui est donc stockée sous la forme d'une suite de nombres binaires, mais qui n'est pas nécessairement un nombre aux yeux du langage de programmation utilisé. Cela peut être en fait à peu près n'importe quel « objet » susceptible d'être placé dans la mémoire d'un ordinateur, par exemple : un nombre entier, un réel, un complexe, un vecteur, une chaîne de caractères typographiques, un tableau, une fonction, ···

Sous *Python*, les noms de variables doivent obéir à quelques règles simples :

- Un nom de variable est une séquence de lettres (de a à z et de A à Z) et de chiffres (0 à 9), qui doit toujours commencer par une lettre.
- Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, %, $\cdots$ sont interdits, à l'exception du caractère _ (souligné donné par la touche 8).
- La casse est significative (les caractères majuscules et minuscules sont distingués). Ainsi, `Toto`, `toto`, `TOTO` et `ToTo` sont donc des variables différentes.

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans `tableDesMatieres`, par exemple.

Évitez si possible les caractères ℓ, 1 et I que l'on peut confondre ainsi que les 0 que l'on peut confondre avec des O.

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme noms de variables les 33 « mots réservés » ci-dessous (ils sont utilisés par le langage lui-même) :

mot	utilisation	page
`and`	opérateur logique `ET`	45
`as`	associé à `import` pour utiliser les bibliothèques	297, 302
`assert`	permet de rajouter des contrôles pour le débogage d'un programme	non étudié
`break`	interrompt une boucle `while` ou `for`	66
`class`	utilisée en programmation orientée objet (POO)	non étudié
`continue`	saute l'étape suivante dans une boucle `while` ou `for`	66
`def`	définit une fonction	68, 136
`del`	supprime un ou plusieurs éléments d'une liste à partir de leur index	53
`elif`	instruction conditionnelle `SINON SI`	61
`else`	instruction conditionnelle `SINON`	61
`except`	gestion d'erreurs	80
`False`	booléen `FAUX`	48
`finally`	gestion d'erreur	80
`for`	boucle `POUR`	63
`from`	importation de bibliothèques	302
`global`	définit des variables globales dans une fonction	74
`if`	instruction conditionnelle `SI`	61
`import`	importation de bibliothèques	302
`in`	test d'appartenance	53
`is`	test d'égalité	49
`lambda`	définit une fonction par une expression	136
`None`	objet qui ne vaut "rien"	70
`nonlocal`	gestion des variables (locales et globales)	74
`not`	opérateur logique `NON`	62
`or`	opérateur logique `OU`	45
`pass`	instruction vide	62
`raise`	levée d'exception	82
`return`	valeur de retour d'une fonction	68
`True`	booléen `VRAI`	48
`try`	gestion d'erreur	80
`while`	boucle `TANT QUE`	65
`with`	simplification d'écriture	non étudié
`yield`	remplace `return` dans un générateur	non étudié

TABLE 5.1 – Mots réservés

Vous retrouverez cette liste en annexe page 388.

Les noms commençant par une majuscule ne sont pas interdits, mais l'usage veut qu'on le réserve plutôt aux variables qui désignent des classes (le concept de classe ne sera pas abordé en CPGE).

Une variable sert d'espace de stockage pour les résultats intermédiaires d'un calcul. Elle possède un certain nombre de caractéristiques :

- **identificateur** : c'est son nom. Il doit être **bien choisi** – surtout dans un programme long – et respecter les règles précédentes.
- **type** : Dans certains langages, le type d'une variable peut changer en cours d'exécution du programme. C'est le cas de *Python*.
- **portée** : On ne peut utiliser le nom d'une variable que dans la fonction ou la procédure (en réalité le bloc) qui la contient. La durée de vie de la variable correspond à la durée d'exécution de la fonction ou de la procédure (du bloc). Sans ce principe, les programmes longs deviendraient incompréhensibles et difficiles à corriger.

⚠ Attention à ne pas confondre le symbole $=$, utilisé comme symbole d'affectation avec le symbole $=$ d'égalité mathématique qui indique que deux valeurs sont égales. L'équation mathématique $a = a + 1$ est par exemple sans intérêt en mathématiques, alors que la ligne de programme $a = a + 1$ indique que la variable a doit être incrémentée de 1.

5.5 Types numériques

En mathématiques, les nombres appartiennent à des ensembles tels que $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$. En *Python* (comme dans beaucoup de langages informatiques) les nombres seront du type :

- `int` (integer) : correspond à un entier ($\mathbb{N}$ ou $\mathbb{Z}$). Sa taille est limitée par la mémoire de l'ordinateur.
- `float` : le type flottant, permet de représenter des nombres à virgule. Il est codé en mémoire sur 8 octets (64 bits). Il est représenté sous la forme nombre $= \pm 1.\text{mantisse} * 2^{\pm \text{exposant}}$. La mantisse est écrite avec 52 chiffres binaires, et l'exposant avec 10 ; il y a deux bits de signe. La précision maximale est donc de l'ordre de 2×10^{-16}. Des valeurs spéciales permettent de représenter $-\infty$, $+\infty$ et `Nan`, *not a number*, souvent issu d'une forme indéterminée..
  Les réels, que l'on manipule en mathématiques, n'existent pas en *Python* : ils sont remplacés par les nombres à virgule flottante. Cela implique que tous les calculs sur ordinateur sont intrinsèquement faux ...
- `complex` : le type complexe correspond à une structure naturellement composée de deux flottants (partie réelle et imaginaire) sur 2 x 8 = 16 octets.
- `bool` : le type booléen correspond à l'algèbre booléenne et ne prend que deux valeurs, `True` / `False`. Il est codé sur un bit.

En pratique, pour des applications basiques, le typage est la plupart du temps transparent pour l'utilisateur car *Python* interprète dynamiquement le type des variables.
Par exemple sur une machine 32 bits, on peut lancer la séquence suivante, illustrant le type *entier int* :

Exemple pour les entiers :

```
>>> c=2**10000
>>> c
199506311688075838488374216268358508382349683188619245485200894985294388302219466319199616840361945978993311294232091242715564913494137811175937859320963239578557300467937945267652465512660598955205500869181933115425086084606181046855090748660896248880904898948380092539416332578506215683094739025569123880652250966438744410467598716269854532228685381616943157756296407628368807607322285350916414761839563814589694638994108409605362678210646214273333940365255656495306031426802349694003359343166514592977732796657756061726820314079941981796073782456837622800373028854872519008344645814546505579296014148339216157345881392570953797691192778008269577356744441230620187578363255027283237892707103738028663930314281332414016241956716905740614196543423246388012488561473052074319922596117962501309928602417083408076059323201612684922884962558413128440615367389514871142563151110897455142033138202029316409575964647560104058458415660720449628670165150619206310041864222759086709005746064178569519114560550682512504060075198422618980592371180544447880729063952425483392219827074044731623767608466130337787060398034131971334936546227005631699374555082417809728109832913144035718775247685098572769379264332215993998768866608083688378380276432827751722736575727447841122943897338108616074232532919748131201976041782819656974758981645312584341359598627841301281854062834766490886905210475808826158239619857701224070443305830758690393196046034049731565832086721059133009037528234155397453943977152574552905102123109473216107534748257407752739863482984983407569379556466386218745694992790165721037013644331358172143117913982229838458473344402709641828510050729277483645505786345011008529878123894739286995408343461588070439591189858151457791771436196987281314594837832020814749821718580113890712282509058268174362205774759214176537156877256149045829049924610286300815355833081301019876758562343435389554091756234008448875261626435686488335194637203772932400944562469232543504006780272738377553764067268986362410374914109667185570507590981002467898801782719259533812824219540283027594084489550146766683896979968862416363133763939037345580140763674187771105538422573949911018646821969658165148513049422236994771476306915546821768287620036277725772378136533161119681128079266948188720129864366076855163986053460229787155751794738524636944692308789426594821700805112032236549628816903573912136833839359175641873385051097027161391543959099159815465441733631165693603112224993796999922678173235802311186264457529913575817500819983923628461524988108896023224436217377161808635701546848405862232979285387562348655644053696262201896357102881236156751254333830327002909766865056855715750551672751889919412971133769014991618131517154400772865057318955745092033018530484711381831540732405331903846208403642176370391155063978900074285367219628090347797453332046836879586858023795221862912008074281955131794815762444829851846150970488802727472157468813159475040973211508049819045580341682694978714131606321068639151168177430479259670937 6
>>> type(c)
<class 'int'>
```

Remarque importante

Si un programme est "planté" ou met trop de temps à s'exécuter, on peut arrêter son exécution par la commande `CTRL - C`.

Pour les booléens (| et & indiquent respectivement *OU* et *ET*), on peut lancer la séquence :

```
>>> a=True
>>> b=False
>>> a|b
True
>>> a&b
False
```

Remarques

Pour les booléens :

- $a|b$ est équivalent à a or b
- $a\&b$ est équivalent à a and b

mais en règle générale, ce n'est pas vrai :
http://python.developpez.com/cours/DiveIntoPython/php/frdiveintopython/power_of_introspection/and_or.php

On peut représenter la table de vérité suivante :

a	b	a and b	a or b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

TABLE 5.2 – table de vérité

Pour les nombres complexes, on peut tester :

```
>>> a=1.5+3j
>>> b=1.5
>>> a-b
3j
>>> a*b
(2.25+4.5j)
>>> c=1+1j
>>> a*c
(-1.5+4.5j)
>>> (1+4j)**(1/(2+3j))
(1.67585169460166-0.20706889169199882j)
>>> 1j * complex(0,1)
(-1+0j)
```

En réalité, il est possible d'utiliser la structure des nombres complexes de façon plus évoluée. Par exemple :

```
>>> a=1.5+3j
>>> a.real
1.5
>>> a.imag
3.0
>>> a.conjugate()
(1.5-3j)
```

Remarque

`a.conjugate` est appelée une fonction ou encore mieux, une méthode.

Enfin, il est possible de modifier explicitement le type des variables en utilisant les fonctions de conversion :

`int(a)`	Conversion de a en type entier
`long(a)`	Conversion a en type entier long [1]
`float(a)`	Conversion a en type flottant
`complex(a)`	Conversion a en type complexe a+0j
`complex(a,b)`	Conversion de a et de b en type complexe a+bj

TABLE 5.3 – Fonctions de conversion de type

Sous *Python*, il existe la fonction `type()` qui donne le type d'une variable. Par exemple :

```
>>> a=1
>>> type(a)
<class 'int'>
>>> a=1.
>>> type(a)
<class 'float'>
>>> a=2**1000
>>> type(a)
<class 'int'>
>>> a=3+4j
>>> type(a)
<class 'complex'>
```

1. Avec *python 2.7* mais pas avec *python 3*

5.6 Opérations sur les nombres

5.6.1 Opérateurs arithmétiques

Python ne possède que très peu d'opérations arithmétiques pour manipuler les nombres. À ce stade, *Python* se résume à une calculatrice élémentaire :

Opération	Signification	Exemple	Résultat
+	Addition	a="Les"+ " " + "chats"	a = "Les chats"
−	Soustraction	b = 8 - 2	b = 6
*	Multiplication	c = "to" * 2	c = "toto"
/	Division	d = 7 / 2	d = 3.5
**	Puissance	e = 2 ** 3	e = 8
%	Reste de division entière	f = 7 % 2	f = 1
//	Quotient de division entière	g = 7 // 2	g = 3

TABLE 5.4 – Opérateurs arithmétiques

La nature des opérations arithmétiques dépend du type des variables.

5.6.2 Raccourcis pour les opérateurs arithmétiques

Dans une séquence de code, on peut affecter une variable à une valeur, effectuer une opération arithmétique sur cette variable et finalement affecter la nouvelle valeur obtenue à la variable. Par exemple :

```
>>> a=1
>>> b=4
>>> a=a+b
>>> a
5
```

En *Python*, il est possible d'écrire l'opération arithmétique de façon plus compacte :

```
>>> a=1
>>> b=4
>>> a+=b
>>> a
5
```

Ceci est également vrai pour les autres opérateurs arithmétiques :

a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b
a**=b	a=a**b
a%=b	a=a%b

TABLE 5.5 – Raccourcis opérateurs

5.6.3 Opérateurs de comparaison

Python peut aussi vérifier si une comparaison entre deux nombres est vraie ou fausse ce qui renvoie une valeur booléenne. Les opérateurs standards de comparaison sont :

<	Plus petit que
>	Plus grand que
<=	Plus petit que ou égal à
>=	Plus grand que ou égal à
==	Égal à
!=	Différent de
is	Est

TABLE 5.6 – Opérateurs de comparaison

Quelques subtilités entre `is` et == :

```
>>> a = [1,2,3,4]
>>> b=a
>>> b is a
True
>>> b == a
True
>>> c = a[:]
>>> c is a
False
>>> c == a
True
```

5.6.4 Quelques fonctions supplémentaires

De base, Python fournit quelques fonctions numériques autres que les opérations arithmétiques :

`abs(a)`	Valeur absolue de a
`max(...)`	Plus grande valeur d'une suite de nombres
`min(...)`	Plus petite valeur d'une suite de nombres
`round(a,n)`	Arrondi de la variable `a` au niveau de la $n^{\text{ième}}$ décimale

TABLE 5.7 – Autres opérateurs arithmétiques

Par exemple :

```
>>> a=-1
>>> abs(a)
1
>>> a=1
>>> b=3.4
>>> c=10
>>> max(a,b,c)
10
>>> min(a,b,c)
1
>>> a=3.141592654
>>> round(a,4)
3.1416
>>> round(a,2)
3.14
>>> round(a,0)
3.0
```

5.7 Opérations sur les bits

Python dispose de 6 opérateurs de base pour agir directement sur les bits :

`&`	ET
`|`	OU
`^`	OU exclusif
$\sim$	Inversion des bits du nombre situé à droite
$\gg$	Décalage d'un bit à droite (division par 2)
$\ll$	Décalage d'un bit à gauche (multiplication par 2)
`is`	Est

TABLE 5.8 – Opérateurs sur les bits

Exemple :

```
>>> 98&54
34
>>> bin(98)
'0b1100010'
>>> bin(54)
'0b110110'
>>> bin(34)
'0b100010'
```

6

Structures algorithmiques

6.1 Compétences exigées

Objectifs

Les principales capacités développées dans cette partie sont les suivantes :

- concevoir l'en-tête (ou la spécification) d'une fonction, puis la fonction elle-même,
- traduire un algorithme dans un langage de programmation,
- gérer efficacement un ensemble de fichiers correspondant à des versions successives d'un fichier source,
- rechercher une information au sein d'une documentation en ligne, analyser des exemples fournis dans cette documentation,
- documenter une fonction, un programme plus complexe.

6.2 Généralités

L'**indentation du code est fondamentale en *Python*** puisqu'elle détermine la structure du code.

Comme déjà signalé auparavant, les mots-clefs et les identifiants sont tous sensibles à la casse des caractères : `foo`, `Foo`, `fOO` et `FOO` sont des variables différentes.

Les caractères suivant un # sont des commentaires, destinés aux autres lecteurs du code source..

- Il est possible d'appeler l'interpréteur *Python* à partir d'un environnement de développement intégré. Sous *spyder*, *geany* et *idle* cela se fait par F5 , sous *Eclipse* par F11 ou CTRL - F11 . Il est possible de préciser les options de ligne de commande.
  On peut préciser les paramètres à passer au script dans le menu d'exécution.
- Pour exécuter un script *exemple.py* à partir d'un terminal on tape simplement : `python exemple.py`
  Sur un système *Unix/Linux*, si la première ligne du script est de la forme `#!/usr/bin/python` ou `#!/usr/bin/python3` et que l'utilisateur a des droits en exécution dessus, il suffit de taper `exemple.py`.
  La deuxième ligne (ou la première si la précédente est absente) `# -*- coding: utf-8 -*-` est particulière : elle indique à la machine virtuelle *Python* que le code source est écrit en *UTF-8*, ce qui permet les caractères accentués.

6.3 Types de données dynamiques

6.3.1 Chaînes de caractères

Le type chaîne de caractères est nommé `str` en *Python* (`str` comme *string*).
Une chaîne est en fait une chaîne de caractères qui est utilisée pour stocker et représenter de l'information textuelle. D'un point de vue fonctionnel, les chaînes peuvent représenter tout ce qui peut être encodé comme du texte.
Un caractère en *Python* est une chaîne d'un caractère. **Les chaînes de caractères sont des séquences non modifiables** : elle répondent aux opérations habituelles mais elle ne peuvent pas être modifiées, elles sont **non mutables**.

Une chaîne de caractères est délimitée par des apostrophes ou des guillemets.. Les deux sont possibles, mais le double guillemet permet d'avoir une apostrophe sans avoir recours au backslash :  .
Le triple double guillemet permet d'entrer une chaîne de caractères sur plusieurs lignes, y compris les caractères de retour de ligne.

On peut effectuer quelques opérations à partir des chaînes, par exemple :

Chaîne	Interprétation
`ch1="`	chaîne vide
`ch2=' '`	chaîne vide
`ch3= "bla"`	double guillemets
`bloc= """...."""`	bloc

À partir de ces chaînes, on peut effectuer par exemple :

```
>>> ch2=' '
>>> ch3="bla"
>>> ch3+ch2+ch3
'bla bla'
>>> ch3*5
'blablablablabla'
```

Illustration du caractère non modifiable ou non-mutable des chaînes :

```
>>> ch='Rello !'
>>> ch[0]='H'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

mais

```
>>> ch='H'+ch[1:]
>>> ch
'Hello !
```

L'ensemble des méthodes pour les chaînes est rassemblée en annexe page 384. Certaines sont utilisées dans l'exemple ci-dessous :

```
>>> x = "physique"
>>> x.upper()
'PHYSIQUE'
>>> x
'physique'
>>> x[0].upper()+x[1:]
'Physique'
>>> x
'physique'
>>> x.capitalize()
'Physique'
>>> matieres="physique chimie informatique"
>>> matieres.capitalize()
'Physique chimie informatique'
>>> matieres.title()
'Physique Chimie Informatique'
>>> matieres.swapcase()
'PHYSIQUE CHIMIE INFORMATIQUE'
>>> matieres.islower()
True
>>> matieres.isupper()
False
>>> matieres.isalpha()
False
>>> matieres.isdigit()
False
>>> matieres.split()
['physique', 'chimie', 'informatique']
>>> matieres.find("for")
18
>>> matieres.find("phy")
0
>>> matieres.find("hys")
1
>>> matieres.replace("i","o")
'physoque chomoe onformatoque'
>>> matieres.count("ique")
2
>>> for mot in matieres.split():
    print(mot)
physique
chimie
informatique
```

On peut découper une chaîne de caractères en mots par la méthode `split()`.

```
>>> chaine="La méthode split est très pratique"
>>> chaine.split()
['La', 'méthode', 'split', 'est', 'très', 'pratique']
```

On peut spécifier un délimiteur à `split` :

```
>>> chaine="La méthode split est très pratique"
>>> chaine.split('r')
['La méthode split est t', 'ès p', 'atique']
```

Il est possible de référencer l'un des caractères en considérant la chaîne comme une liste (`chaine[3]` pour le 4^e caractère), mais chaque caractère est lui-même de type *str*.

```
>>> chaine="La méthode split est très pratique"
>>> chaine
'La méthode split est très pratique'
>>> chaine[3]
'm'
>>> print(type(chaine[3]))
<class 'str'>
```

Des outils plus évolués de manipulation de chaînes de caractères sont accessibles par le module `re` (cf. paragaraphe 18.2 page 303).

6.3.2 Listes

Liste

Une liste permet de stocker des successions de valeurs, et de les retrouver par leur index dans sa structure. Elle peut être modifiée : modification des éléments présents, suppression, et ajout d'élément(s).

Une `liste` s'écrit entre crochets : `tab = [6, 7]`.

La numérotation des cases commence à 0. L'index se place entre crochets : `tab[4]`. Il est possible de sélectionner des tranches de ces structures : `tab[1:4]`, `tab[:5]`, `tab[3:]`, `tab[1:-1]`. Le premier élément est inclus, le dernier est exclu : `tab[1:3]` contient les cases 1 et 2. L'élément d'indice `-1` est le dernier élément de la liste, l'élément d'indice `-2` le pénultième, ...

Il n'est pas possible de créer une nouvelle case à une liste par une simple affectation. Si `tab` est de taille 6, `tab[8]=42` entraîne une erreur. Il faut ajouter la nouvelle valeur par `tab.append(42)`. On peut concaténer `liste2` à `liste1` par `liste1.extend(liste2)`.

L'opérateur `len()` donne le nombre d'éléments de la liste : `len([3, 7, [5, 3], "Last"])` renvoie 4.

On peut supprimer un élément d'une liste en connaissant son index `i` par `del tab[i]`. On peut supprimer le premier élément ayant une certaine valeur `val` par `tab.remove(val)`.

⚠ `tab.remove(tab[i])` ne supprime donc pas forcément l'élément d'index `i`.

La méthode `pop()` supprime et renvoie le dernier élément d'une liste :

```
>>> liste=[9,6,42]
>>> x=liste.pop()
>>> liste
[9, 6]
>>> x
42
```

La variable `x` vaut désormais 42, et `liste` vaut [9, 6].

La méthode `insert(i, x)` insère l'élément `x` à l'indice `i`. Si `i` est supérieur ou égal à la taille de la liste, `x` est inséré en dernière position, comme un appel à `append(x)`.

```
>>> liste = [9, 6, 42]
>>> liste.insert(1, 7)
>>> liste.insert(100, 24)
>>> liste
[9, 7, 6, 42, 24]
```

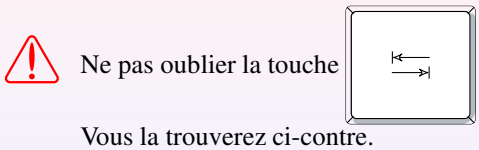
La variable `liste` vaut désormais `[9, 7, 6, 42, 24]`.

On peut tester l'appartenance d'un élément à une liste par l'opérateur `in` .

Remarque

Il est possible d'inverser le comportement de `in` en le remplaçant par `not in`.

```
>>> premiers = [2, 3, 5, 7, 11, 13, 17, 19]
>>> if 5 in premiers:
...     print ("%i est un nombre premier" %5)

5 est un nombre premier
>>> if 4 not in premiers:
...     print("%i n'est pas un nombre premier" %4)

4 n'est pas un nombre premier
```



Un tableau à n dimensions n'est rien d'autre qu'une liste de listes qui sont toutes de la même longueur.

Pour faire une liste de n listes vides, il ne faut pas écrire `tab = n*[[]]`, car on aurait n copies de la même liste :

```
>>> tab = 4*[[]]
>>> tab[1].append(42)
>>> tab
[[42], [42], [42], [42]]
```

mais :

```
>>> tab=[]
>>> for i in range(4):
...     tab.append([])
...
>>> tab[1].append(42)
>>> tab
[[], [42], [], []]
```

Une variable de type `list` référence une zone mémoire. Si deux variables référencent une même zone mémoire, modifier l'une impacte l'autre :

```
>>> liste1 = ["Terre", 245, "Lune"]
>>> liste2 = liste1
>>> liste1.insert(2,"Vénus")
>>> liste1
['Terre', 245, 'Vénus', 'Lune']
>>> liste2
['Terre', 245, 'Vénus', 'Lune']
```

Extraire une tranche d'une liste en réalise une copie d'une partie de son contenu (éventuellement la liste entière). Ce comportement est différent des tableaux de *numpy* (cf. paragraphe 20.4.3 page 324).

```
>>> liste1 = ["Terre", 245, "Lune"]
>>> liste2 = liste1[:]
>>> liste1.insert(2,"Vénus")
>>> liste1
['Terre', 245, 'Vénus', 'Lune']
>>> liste2
['Terre', 245, 'Lune']
```

On peut aussi effectuer une copie de liste avec la fonction `list` :

```
>>> liste1 = ["Terre", 245, "Lune"]
>>> liste2 = list(liste1)
>>> liste1.insert(2,"Vénus")
>>> liste1
['Terre', 245, 'Vénus', 'Lune']
>>> liste2
['Terre', 245, 'Lune']
```

L'ensemble des méthodes pour les listes est rassemblé en annexe page 385.

⇒ **Activité 6.18**

Indiquez les résultats donnés par l'exécution des codes suivants :

1 -

```
>>> list1=["maths","physique","chimie"]
>>> list1.sort()
>>> list1.reverse()
>>> list1
```

2 -

```
>>> list2=["maths","français","anglais","méca","élec","physique"]
>>> list2.pop(3)
>>> list2
>>> len(list2)
>>> list2[0:4:2]
```

3 -

```
>>> list3=list(range(2,19,2))
>>> list4=list3
>>> del(list4[3])
>>> list3.reverse()
>>> list4.remove(10)
>>> list3
>>> list4
```

4 -

```
>>> phrase = ['Python ','c',"'",'est ','du ','béton !']
>>> for mot in phrase:
    print(mot, end ='')
```

Il est possible de récupérer une liste grâce à la suite d'instructions `list(eval())`. Les éléments doivent être séparés par des virgules :

```
>>> ch = input("Votre réponse : ")
Votre réponse : 3,5,6
>>> list(eval(ch))
[3, 5, 6]
```

6.3.3 Tuples

Tuple

Un tuple permet de stocker des successions de valeurs, et de les retrouver par leur index dans la structure, mais ne peut pas être modifié : on dit qu'il est *non-mutable*.

Un `tuple` s'écrit entre parenthèses : `tup = (6, 7)`. Il peut contenir tout type de données non-mutable, y compris d'autres tuples : `tup = (6, 7.05, "poisson", (-5, "dauphin"))`.

On accède aux éléments d'un tuple par leurs index ou leurs tranches d'index exactement comme pour une liste. La fonction `len()` et l'opérateur `in` s'appliquent également.

Il n'est pas forcément nécessaire d'utiliser les tuples mais comme certaines méthodes ou fonctions renvoient des tuples, il faut connaître leurs propriétés.

L'ensemble des méthodes pour les tuples est rassemblé en annexe page 385.

6.3.4 Dictionnaires

Le dictionnaire est un système très souple pour intégrer des données. Si on pense aux listes comme une collection d'objets ordonnés et portant un indice par rapport à la position, un dictionnaire est une liste comportant à la place de ces indices, un mot servant de clé pour retrouver l'objet voulu.
Un dictionnaire est affecté par une paire d'accolades ({ }), par exemple :

```
>>> capitales = {'France' : 'Paris', 'Espagne' : 'Madrid'}
```

Les clés peuvent être de toutes les sortes d'objet non modifiables. Par contre la valeur stockée peut être de n'importe quel type.
Un dictionnaire n'est pas ordonné comme une liste, mais c'est une représentation plus symbolique de classement par clés. La taille d'un dictionnaire peut varier, et le dictionnaire est un type modifiable ou mutable.

Opération	Interprétation
`d1 = {}`	Dictionnaire vide
`d2={'one' : 1, 'two' : 2}`	Dictionnaire à deux éléments
`d3= {'count ': {'one': 1, 'two': 2}}`	Inclusion
`d2 ['one'], d3['count']['one']`	Indiçage par clé
`d2.has_keys('one')`	Méthode : test d'appartenance
`d2.keys()`	liste des clés
`d2.values()`	Liste des valeurs
`len(d1)`	Longueur (nombre d'entrées)
`d2[cle] = [nouveau]`	ajout ou modification
`del d2[cle]`	destruction

TABLE 6.1 – Dictionnaires

Exemple plus complet :

```
>>> tel = { 'Pierre' : 9876, 'Paul': 8765, 'Jacques' : 7654 }
>>> print(tel)
{'Paul': 8765, 'Pierre': 9876, 'Jacques': 7654}
>>> print(tel['Paul'])
8765
>>> tel['Henri']=6543
>>> print(tel)
{'Paul': 8765, 'Pierre': 9876, 'Henri': 6543, 'Jacques': 7654}
>>> print(tel.keys())
dict_keys(['Paul', 'Pierre', 'Henri', 'Jacques'])
>>> print(tel.values())
dict_values([8765, 9876, 6543, 7654])
>>> print(tel.items())
dict_items([('Paul', 8765), ('Pierre', 9876), ('Henri', 6543), ('Jacques', 7654)])
>>> print(len(tel))
4
>>> del tel['Paul']
>>> print(len(tel))
3
```

6.3.5 Les ensembles

Les ensembles en python se définissent grâce à la commande *set*. On distingue 2 types d'ensembles :

- les `set` : modifiables,
- les `frozenset` : non modifiables.

Certaines opérations sont très pratiques.

Définissons tout d'abord 3 ensembles :

```
>>> ens1=set([1,2,3,4,5,6,7,8,9])
>>> ens2=set([2,3,4,5])
>>> ens3=set([5,6])
```

6.3.5.1 Inclusion

L'inclusion, notée en maths $\subset$ peut se faire avec la méthode `issubset` :

```
>>> ens2.issubset(ens1)
True
```

6.3.5.2 Réunion

La réunion, notée en maths $\cup$ peut se faire avec la méthode `union()`, ou avec l'opérateur `|` :

```
>>> ens2.union(ens3)
{2, 3, 4, 5, 6}
>>> ens2 | ens3
{2, 3, 4, 5, 6}
```

6.3.5.3 Intersection

L'intersection, notée en maths ∩ peut se faire avec la méthode `intersection`, ou avec l'opérateur & :

```
>>> ens1 & ens2
{2, 3, 4, 5}
>>> ens1 & ens3
{5, 6}
>>> ens2 & ens3
{5}
>>> ens2.intersection(ens3)
{5}
```

6.3.5.4 Différence

La différence, notée en maths \ peut se faire avec la méthode `difference()`, ou avec l'opérateur − :

```
>>> ens1 - ens2
{8, 1, 9, 6, 7}
>>> ens1.difference(ens2)
{8, 1, 9, 6, 7}
```

6.3.5.5 Différence symétrique

La différence symétrique, notée en maths Δ peut se faire avec la méthode *symmetric_difference()*, ou avec l'opérateur ^. Cette différence symétrique correspond à l'union moins les éléments communs.

```
>>> ens2^ens3
{2, 3, 4, 6}
>>> ens2.symmetric_difference(ens3)
{2, 3, 4, 6}
```

6.4 Affichage

6.4.1 La fonction *input*

Pour réaliser une saisie à l'écran, la fonction *input()* est tout indiquée : elle interrompt le programme, affiche une éventuelle invite et attend que l'utilisateur entre une donnée et la valide par *Entrée* : ENTER .

⚠ La fonction `input()` effectue toujours une saisie en mode texte (la saisie est alors une chaîne) dont on peut ensuite changer le type (on dit aussi transtyper).

```
>>> age=input("Entrez votre age : ")
Entrez votre age : 25
>>> print(type(age))
<class 'str'>
```

Il faut alors transtyper la variable *age* :

```
>>> age=int(age)
>>> print(type(age))
<class 'int'>
```

ou plus directement :

```
>>> age=int(input("Entrez votre age : "))
Entrez votre age : 25
>>> print(type(age))
<class 'int'>
```

6.4.2 La fonction *print*

La fonction `print()` permet l'écriture sur la sortie standard. Lors d'un appel à `print(x)`, si x n'est pas une chaîne de caractères, `x` est converti en chaîne de caractères par `str(x)`.

Il est possible de demander l'affichage de plusieurs objets en les séparant par des virgules :

```
>>> print(6, "x", 7, " = ",6*7)
6 x 7  =  42
```

Si le dernier objet est suivi de *end=''*, il n'y a pas de retour à la ligne (Attention, le retour à la ligne est évité en mettant une virgule à la fin).

On peut choisir de mettre un espace *end=' '* ou tout autre caractère.

```
>>> print(6, "x", 7, " = ",6*7, end='------------\n')
6 x 7  =  42------------
```

Certains caractères spéciaux, comme les tabulations ou les retours à la ligne, sont codés par caractères échappés : `'\t'` pour une tabulation, `'\n'` pour un retour à la ligne. Il faut échapper l'anti-slash par `'\\'` pour l'inclure dans une chaîne de caractères.

Ce caractère antislash \ permet de donner une signification spéciale à certaines séquences :

Séquence	Signification
\	saut de ligne ignoré
\\	antislash
\'	apostrophe
\"	guillemets
\a	sonnerie (bip)
\b	retour arrière
\f	saut de page
\n	saut de ligne
\r	retour début de ligne
\t	tabulation horizontale
\v	tabulation verticale
\N{nom}	caractère sous forme Unicode avec nom
\uhhhh	caractère sous forme Unicode 16 bits
\uhhhhhhhh	caractère sous forme Unicode 32 bits
\ooo	caractère sous forme de code octal
\xhh	caractère sous forme de code hexadécimal

TABLE 6.2 – Séquences d'échappement

On pourra par exemple étudier les instructions et résultats suivants :

```
>>> table = " Nom\t| Age\nMar\\/in\t|60E9"
>>> print (table)
 Nom     | Age
Mar\/in |60E9
```

On peut demander à *Python* de ne pas interpréter les échappements dans une chaîne en la précédant par `r`.

```
>>> table = r" Nom\t| Age\nMar\\/in\t|60E9"
>>> print (table)
 Nom\t| Age\nMar\\/in\t|60E9
```

6.4.3 Opérateur %

Opérateur %

L'opérateur % permet de placer une ou plusieurs valeurs dans une chaîne.

```
>>> masse=65
>>> qte=3
>>> masse_mol=masse/qte
>>> print("Masse=  %i, quantité de matière= %i,\
masse molaire = %.3f" %(masse,qte,masse_mol))
Masse=  65, quantité de matière= 3, masse molaire = 21.667
```

Il doit y avoir autant d'éléments dans le tuple de valeurs à insérer que de caractères % dans la chaîne de caractères. La lettre qui suit le caractère % dans la chaîne détermine le type d'information à écrire : `i` pour un nombre entier ou `d` pour le nombre de chiffres avant la virgule d'un nombre entier ou réel ; `x` pour un nombre entier hexadécimal ; `f`, `e`, `g`, `E` ou `G` pour un nombre à virgule flottante ; `s` pour une chaîne de caractères, ...
Les nombres qui précèdent permettent de maîtriser le nombre de caractères à écrire, et notamment la précision d'une nombre à virgule flottante.

Un exemple assez utile en sciences :

```
>>> a = 12369872.2587593
>>> print("%2.3e" %a)
1.237e+07
```

6.4.4 Le formatage

Opérateur format

Cette fonction permet d'imprimer les données avec un nombre spécifique de décimales, à la manière de l'opérateur %.

```
>>> masse=2.45
>>> volume=5.9
>>> m_v=masse/volume
>>> print("Masse volumique=",m_v)
Masse volumique= 0.4152542372881356
>>> print("La masse volumique vaut {:.3f} kg/m^3".format(m_v))
La masse volumique vaut 0.415 kg/m^3
>>> print("La masse vaut : {},\nLe volume vaut : {:.2f},\n {}\
\n\tLa masse volumique vaut : {:.4f}".format(masse,volume,"-"*50,m_v))
La masse vaut : 2.45,
Le volume vaut : 5.90,
--------------------------------------------------
    La masse volumique vaut : 0.4153
```

La dernière notation permet d'arrondir les valeurs au nombre de décimales voulu, d'insérer des caractères (par exemple, des tirets ici).

Rappel

Les instructions "\n" et "\t" permettent respectivement de passer à la ligne et d'insérer une tabulation horizontale.

6.4.5 Résumé

En résumé, on pourra retenir les façons suivantes, illustrées dans le programme suivant pour afficher des données :

formatage.py

```
1  # -*- coding: utf-8 -*-
2
3  print("Vous avez %i euros" % 15)
4  a = 20
5  print("Vous avez %i euros" % a)
6  print("Vous avez %i %s" % (a, "euros"))
7  print("Vous avez {} euros".format(a))
8  print("Vous avez {0} euros".format(a))
9  print("Vous avez {1} {0}".format("euros", a))
10 print("Vous avez %.2f euros" %a)
11 print("Vous avez %1.2e euros" %a)
```

Ce programme donne :

```
Vous avez 15 euros
Vous avez 20 euros
Vous avez 20 euros
Vous avez 20 euros
Vous avez 20 euros
Vous avez 20 euros
Vous avez 20.00 euros
Vous avez 2.00e+01 euros
```

6.5 Boucles et conditionnelles

6.5.1 Conditionnelle

Le branchement s'écrit :

```
if condition 1:
    instructions 1
elif condition 2:
    instructions 2
else:
    instructions 3
```

Programme *condition.py* :

condition.py

```
1  # -*- coding: utf-8 -*-
2
3  x = 42
4  if x != int(x):
```

```
    print ("Non entier")
elif x<0:
    print ("Entier négatif")
else:
    print ("Entier positif")
```

Entre `if` et les deux-points, il est possible de mettre toute expression booléenne, issue par exemple d'un opérateur de comparaison comme ==, <, >, <=, >= ou !=. Notez que l'opérateur de test d'égalité est == alors que l'opérateur d'affectation est =. Le test de non-égalité s'écrit!=. L'opérateur de test d'appartenance `in` construit également une expression booléenne.

Il est possible de combiner des expressions booléennes par `and`, `or` et `not`.

On peut aussi écrire une condition de façon plus compacte si le code est simple :

condition-compacte.py

```
# -*- coding: utf-8 -*-

print("Entrez 2 entiers :")
x=int(input("x ? "))
y=int(input("y ? "))

# méthode classique :
if x < y:
    petit = x
else:
    petit = y

# méthode compacte :
petit = x if x < y else y

print("\nLe plus petit est", petit)
```

pass

L'instruction `pass` ne fait rien, et permet éventuellement de remplir une obligation syntaxique (*Python* n'accepte pas les blocs vides).

```
if x<0 or x>=20:
    pass
else:
    print ("%i possède une valeur comprise dans l'intervalle\
    [0, 20[" %x)
```

complémentaire

Notez que prendre le complémentaire de l'expression booléenne fonctionne aussi, et est plus élégant :

```
if not(x<0 or x>=20):
    print ("%i possède une valeur comprise dans l'intervalle\
    [0, 20[" %x)
```

ou encore :

```
if x>=0 and x<20:
    print ("%i possède une valeur comprise dans l'intervalle\
    [0, 20[" %x)
```

Il n'existe pas d'équivalent du `switch ... case` courant dans d'autres langages. Il convient d'utiliser une succession de `elif` pour choisir entre de multiples actions possibles en fonction de la valeur d'une variable.

⇒ **Activité 6.19**

Écrivez un programme qui demande des notes (par exemple Maths, Informatique, Physique et Chimie (le reste étant quantité négligeable ☺) puis calcule la moyenne de ces notes et affiche alors "Menteur", "Félicitations", "Mention Très Bien", "Bien", "Assez Bien", "Passable", "Rattrapage", "Recalé" suivant que la moyenne est supérieure à 20, comprise entre 18 et 20, comprise entre 16 et 18, comprise entre 14 et 16, entre 12 et 14, entre 10 et 12, entre 8 et 10, inférieure à 8.

Le programme :

6.5.2 Type booléen

Il est également possible d'affecter la valeur d'une expression booléenne à une variable, lequel est donc de type `bool`, et peut prendre uniquement les deux valeurs `True` et `False`.

```
>>> a=2
>>> b=3
>>> print(a==b)
False
>>> print(True or False)
True
>>> print(True and False)
False
>>> maliste=["As", "Valet", 10]
>>> "As" in maliste
True
```

6.5.3 Boucles *for* et *while*

6.5.3.1 Boucle *for*

La boucle `for` s'écrit :

```
for var in table:
    instructions
```

Le corps de la boucle est exécuté une fois pour chaque élément de `tab`, lequel peut être tout objet contenant une succession d'éléments. Il peut donc s'agir d'une liste, mais aussi d'une table de hachage, d'un tuple, d'un fichier, ou d'une fonction itératrice comme `range()`.

La fonction `range()` permet (entre autres) de générer une succession d'indices pour `for`, mais aussi toute succession de nombres : `range(début, fin, pas)`.

Ainsi, l'exemple suivant comporte deux boucles. `x` prend successivement les valeurs contenues dans les cases de `tab`. L'indice `i` prend successivement toutes les valeurs entières de 0 à `len(tab)-1` puis `len(tab)-3`, c'est-à-dire les valeurs d'index valides pour `tab`.

boucles.py

```
1  # -*- coding: utf-8 -*-
2
3  tab = [6, 7, 2, 8, 42, 14]
4
5  for i in range(len(tab)):
6      print (i, tab[i])
7
8  print()
9
10 for i in range(len(tab)-2):
11     print (i, tab[i])
```

⇒ **Activité 6. 20**

Affichez les entiers de 0 à 31 (31 non compris), de trois en trois, en utilisant une boucle `for` et l'instruction `range()`.

Le programme :

⇒ **Activité 6. 21**

Affichez chaque caractère de la chaîne "bonjour" en utilisant une boucle `for`. Affichez chaque élément de la liste $[$"*hello*", [2,6], 3.1416$]$ en utilisant une boucle `for`.

Le programme :

6.5.3.2 Boucle *while*

La boucle `while` s'écrit :

```
while cond:
    instructions
```

Le corps de la boucle est exécuté chaque fois que la condition est vraie, et la boucle s'arrête dès que la condition est testée comme fausse. Il est donc possible que le corps de la boucle ne soit jamais exécuté.

⇒ **Activité 6.22**

À l'aide d'une boucle `while`, calculez la somme d'une suite de nombres non nuls entrés par l'utilisateur.
Comptez combien il y a de données et combien sont supérieures à 100.
Un nombre égal à 0 indique la fin de la boucle.

Le programme :

⇒ **Activité 6.23**

Écrivez, à l'aide d'une boucle `while not`, un programme qui demande à l'utilisateur d'entrer un entier pair et qui ne s'arrête que lorsque c'est le cas.

Le programme :

6.5.4 Boucle définissant une liste

Il est immédiat de faire une boucle qui remplit une liste par des appels successifs à `append()` :

```
>>> y = []
>>> for x in range(-10, 10, 1):
    y.append(x**3)
```

Toutefois ces appels à `append()` sont assez coûteux en temps de calcul, et la syntaxe est assez lourde. *Python* propose une alternative plaçant `for` à l'intérieur de la définition de la liste (cf. page 77).

```
>>> y = [x**3 for x in range(-10, 10, 1)]
```

6.5.5 break et continue

break et continue

Dans les boucles, l'instruction `break` sort de la plus petite boucle `for` ou `while` englobante.
L'instruction `continue`, continue sur la prochaine itération de la boucle.

Exemples :

break_continue.py

```
1  # -*- coding: utf-8 -*-
2
3  tab = [6, 7, 2, 8, 42, 14, 12, 21]
4  for x in tab:
5      if 42%x!=0:
6          continue
7      print (x, end=' ')
8  print()
9
10 i = 0
11 while True: # ou while 1:
12     if i==len(tab)-4:
13         break
14     print ("position",i,"-> élément",tab[i])
15     i += 1
```

Les instructions de boucle ont une clause `else` ; elle est exécutée lorsque la boucle se termine par épuisement de la liste (avec `for`) ou quand la condition devient fausse (avec `while`), mais pas quand la boucle est interrompue par une instruction `break`. Ceci est expliqué dans la boucle suivante, qui recherche des nombres premiers :

break-e.py

```
1  # -*- coding: utf-8 -*-
2
3  for n in range(2, 10): # pour n compris entre 2 et 9 inclus
4      for x in range(2, n): # pour x compris entre 2 et n-1 inclus
5          if n % x == 0: # si le reste de n/x = 0 alors
6              print (n, "égale", x, "* {}".format(int(n/x)))
7              break
8      else:
9              print (n, "est un nombre premier")
```

qui donne :

```
2 est un nombre premier
3 est un nombre premier
4 égale 2 * 2
5 est un nombre premier
6 égale 2 * 3
7 est un nombre premier
8 égale 2 * 4
9 égale 3 * 3
```

⇒ **Activité 6.24**

Indiquez le résultat de la suite d'instructions suivante :

break-act.py

```
1  # -*- coding: utf-8 -*-
2
3  for x in range(1, 11): # pour x compris entre 1 et 10 inclus
4      if x == 5: # si x = 5 alors
5          break
6      print(x)
7
8  print("Boucle terminée si x = ", x)
```

⇒ **Activité 6.25**

Indiquez le résultat de la suite d'instructions suivante :

continue-act.py

```
1  # -*- coding: utf-8 -*-
2
3  for x in range(1, 6): # pour compris entre 1 et 5 inclus
4      if x == 2: # si x = 2 alors
5          continue
6      print(x)
7
8  print("Le 2 est passé")
```

⇒ **Activité 6.26**

Indiquez le résultat de la suite d'instructions suivante :

break.py

```
1  # -*- coding: utf-8 -*-
2
3  while 1: # 1 est toujours vrai -> boucle infinie
4      lettre = input("Tapez 'Q' pour quitter : ")
5      if lettre == "Q":
6          print("Fin de la boucle")
7          break
```

6.6 Fonctions et procédures

Les fonctions et procédures sont primordiales pour la programmation procédurale. Il faut :

- comprendre pourquoi on écrit des fonctions,
- s'obliger à le faire dès les premiers programmes.

La différence essentielle entre les deux, c'est que :

- Une fonction effectue un calcul et vaut quelque chose.
- À l'inverse, une procédure n'a pas de valeur mais effectue une tâche (afficher, modifier un objet par exemple).

6.6.1 Notion de fonction

Il est bien sûr possible de créer ses propres fonctions, qui reçoivent éventuellement des paramètres à traiter, et renvoient un résultat via `return`.

La fonction se déclare avec le mot-clé `def`.

Il faudra distinguer la définition de la fonction et son utilisation :

- définition : on indique comment "marche" la fonction et comment on s'en sert (déclaration). La fonction n'est pas utilisée (exécutée), mais juste "lue" pour être connue de l'interpréteur.
- utilisation : on exécute de manière effective le code de la fonction pour des valeurs d'entrées fixées.

⇒ **Activité 6. 27**

Écrivez une fonction qui calcule le montant *TTC* à partir d'un montant *HT*.

Remarque

L'instruction `return` termine la fonction, même si cette instruction est exécutée avant la fin du texte de la fonction.

Un autre exemple de fonction :

racines2.py

```
1  # -*- coding: utf-8 -*-
2
3  def discr(a, b, c):
4      return b**2-4*a*c
5
6  def racines(a, b, c):
7      delta = discr(a, b, c)
8      if delta > 0:
9          return (-b-delta**0.5)/(2*a), (-b+delta**0.5)/(2*a)
10     elif delta == 0:
11         return -b/(2*a),
12     else:
13         return("racines complexes")
14
15 print(racines(2,3,5))
16 print(racines(2,7,5))
```

⇒ **Activité 6. 28**

Déterminez le rôle du code précédent.

Voici quelques occasions dans lesquelles vous devez écrire des fonctions (ou des procédures) :

- Le bloc de programme que vous écrivez ne tient pas en entier à la vue sur votre écran.
- Vous utilisez plus d'une fois des portions de code exactement identiques, ou presque identiques. Vous gagnerez en clarté et en investissement à écrire une fonction à la place de ce bloc et à l'appeler plusieurs fois.
- Vous ne savez pas exactement si la méthode de résolution employée pour telle tâche est la meilleure. Mettez la dans une fonction et utilisez cette fonction. Si vous trouvez une meilleure façon de procéder, il suffira de modifier la fonction sans toucher au reste.

- Le problème que vous traitez est difficile : découpez-le en fonctions que vous écrirez et testerez séparément les unes des autres (le bénéfice que vous aurez à pouvoir tester très facilement des portions de programme est énorme). Dans chaque fonction indépendante, vous pourrez choisir des noms de variables appropriés, qui rendront votre code plus simple à comprendre.

⇒ **Activité 6. 29**

Écrivez, à l'aide d'une boucle `for`, une fonction qui renvoie la factorielle d'un nombre quelconque.

Le programme :

Il est parfois intéressant d'utiliser le mot réservé `None` afin d'affecter une valeur qui ne vaut rien. Un exemple ci-dessous :

none.py

```
1  # -*- coding: utf-8 -*-
2
3  maliste=[1,2,3]
4
5  def test(x,a):
6      if a in x:
7          return("Yes")
8      else:
9          return(None)
10
11 print(test(maliste,3))
12 print(test(maliste,4))
```

6.6.2 Notion de procédure

En *Python*, il n'y a pas de différence syntaxique entre fonction et procédure mais il ne faut pas les confondre.

Une procédure a un effet de bord : elle modifie quelque chose qui est en dehors de la procédure.
En principe, on souhaite qu'une fonction n'ait pas d'effet de bord et c'est pour cette raison qu'on évite si possible les affichages dans une fonction.

Voici la procédure qui calcule le montant TTC à partir d'un montant HT :

proc-ttc.py

```
1  # -*- coding: utf-8 -*-
2
3  def proc_ttc(valht,taux):
```

```
4       """
5       Affiche le montant TTC, connaissant le montant HT (valht)
6       et le taux de TVA (taux)"""
7       valttc=valht*(1+taux/100)
8       print(valttc)
9
10  proc_ttc(120,4)
```

On pourrait penser que la fonction *fonc-ttc* et la procédure *proc-ttc* sont équivalentes. Ce serait une erreur. La fonction est plus générale. On ne peut pas écrire par exemple :

```
>>> print("Double du prix ttc : ",proc-ttc(100,19.6)*2)
```

En revanche, la même ligne, en utilisant *fonc-ttc* fournirait le bon résultat. Assurez-vous d'avoir bien compris pourquoi la ligne qui précède marche avec une fonction et pas avec une procédure. C'est vraiment très important !

⇒ **Activité 6.30**

Écrivez une procédure qui écrit la table de multiplication d'un nombre qu'on appellera $base$, qui commence à $debut$, qui s'arrête à fin par pas de pas.

Le programme :

6.6.3 Documentation

⚠ Il est très important de documenter les fonctions que vous allez créer, pour vous ou pour d'autres qui vont les utiliser. Pour cela, c'est très simple, il suffit de placer immédiatement sous la ligne de définition de votre fonction, une chaîne de caractères entre triple 'quote' (""" ou ''') qui sera considérée un commentaire. On parle aussi de *docstring*.

Exemple :

```
def fonction(argument1, argument2) :
    """
    Ceci est une fonction qui ne fait rien mais qui prend
    2 arguments : argument1 et argument2
    """
    pass
```

6.6.4 Variables locales et globales

Si on affecte une valeur à une variable dans la fonction ou dans la procédure, on définit une variable *locale*. Cette variable n'existe qu'à l'intérieur de la fonction, et n'est plus accessible ensuite. Si une fonction f_1 a une variable locale x et appelle une fonction f_2, la variable x n'est pas disponible dans f_2.

Les paramètres de la fonction se comportent comme des variables locales à cette fonction.

Si une variable existe déjà à l'extérieur de la méthode avec le même nom — c'est donc une variable *globale* — la variable locale masque la variable globale. Les opérations réalisées à l'intérieur de la fonction portent uniquement sur la variable locale, et la variable globale de même nom n'est pas modifiée.

⇒ **Activité 6.31**

Interprétez le résultat des programmes suivants :

1. Programme *variable1.py*.

variable1.py

```
1  # -*- coding: utf-8 -*-
2
3  x = 2
4  def f(x):
5      x = x + 10
6      return x
7  print(f(x))
8  print(x)
```

2. Programme *variable2.py*.

variable2.py

```
1  # -*- coding: utf-8 -*-
2
3  x = 2
4  def f(y):
5      y = y + 10
6      return y
7  print(f(x))
8  print(x)
```

3. Programme *variable3.py*.

variable3.py

```
1  # -*- coding: utf-8 -*-
2
3  x = 2
4  def f(y):
```

```
5       y = y + 10
6       return x
7   print(f(x))
8   print(x)
```

4. Programme *variable4.py*.

variable4.py

```
1   # -*- coding: utf-8 -*-
2
3   x = 2
4   def f(y):
5       y = y + 10
6       return y
7   print(f(x))
8   print(y)
9   print(type(y))
```

Si on souhaite modifier une variable globale à l'intérieur d'une fonction, il convient de faire référence à cette variable globale par `global`.

⇒ **Activité 6.32**

Indiquez le résultat donné par l'exécution du code suivant :

variables.py

```
1   # -*- coding: utf-8 -*-
2
3   def f_loc(x):
4       a = x
5       print ("f_loc : \t", a)
```

```
6
7  def f_glob(x):
8      global a
9      a = x
10     print ("f_glob : \t", a)
11
12 def g(x):
13     print ("g : \t\t", a)
14
15 a = 42
16 f_loc(33)
17 g(24)
18 f_glob(65)
19 f_loc(76)
20 g(77)
```

6.6.5 Local, englobant, global et built-in

Les auteurs anglophones conseillent de retenir la règle "legb", acronyme qui permet de mémoriser l'ordre de résolution des noms de variable dans *Python* : *local*, *englobant*, *global* et *built-in*.

- Les variables *locales* à une fonction sont celles définies dans cette dernière ainsi que ses paramètres. **Elles n'existent donc pas en dehors de cette fonction**,
- Les variables du bloc englobant d'une fonction sont celles dont la portée contient celle-ci,
- Les variables *globales* sont celles déclarées au niveau du fichier python,
- Enfin, *built-in* regroupe ce qui est défini à l'exécution de l'interpréteur.

Remarque

Dans un premier temps, il est préférable de ne pas utiliser de variable globale, ce qui est une bonne chose de toutes façons.

6.6.6 Fonction retournant plusieurs valeurs

Une fonction est capable de retourner plusieurs valeurs en une seule.
Ces valeurs peuvent être récupérées dans différentes variables ou dans un tuple.

Ainsi, le programme suivant :

fonc_var_mul.py

```
1  #-*- coding: utf-8 -*-
2
3  def maFonction(a,b,c):
4      return (a+b,b+c,a-c)
5
6  print(maFonction(10,5,3))
```

```
7
8  # récupération dans des variables
9  aa, bb, cc = maFonction(10,5,3)
10 print( "aa vaut %i, bb vaut %i, cc vaut %i" % (aa, bb, cc) )
11
12 # récupération via un Tuple
13 resultat = maFonction(10,5,3)
14 print( "aa est égal à %i, bb est égal à %i, cc est égal à %i" %
                                          resultat )
```

retourne :

```
(15, 8, 7)
aa vaut 15, bb vaut 8, cc vaut 7
aa est égal à 15, bb est égal à 8, cc est égal à 7
```

Si `return` est suivi par plusieurs paramètres, ceux-ci sont placés dans un tuple. S'il n'y a pas de `return` (cas où $\Delta < 0$ dans le programme *racines*), la valeur renvoyée est la valeur spéciale [1] `None` de type `NoneType`.

Si on souhaite garantir que *racines* renvoie toujours un tuple, même vide, il faut traiter le cas $\Delta < 0$, et forcer la création d'un tuple quand $\Delta = 0$, par l'ajout d'une virgule ou la création explicite d'un tuple avec `return tuple([-b/2/a])`.

racines2-t.py

```
1  # -*- coding: utf-8 -*-
2
3  # Détermine le discriminant de a x^2 + b x + c = 0
4  def discr(a, b, c):
5      return b**2-4*a*c
6
7  """ Détermine les racines de a x^2 + b x + c = 0 """
8  """ Renvoie un tuple """
9
10 def racines(a, b, c):
11     delta = discr(a, b, c)
12     if delta > 0:
13         return (-b-delta**0.5)/2/a, (-b+delta**0.5)/2/a
14     elif delta == 0:
15         return -b/2/a,
16     else:
17         return ()
```

6.6.7 Arguments d'une fonction

Les fonctions *Python* :

- acceptent des valeurs par défaut,
- permettent de retourner plusieurs valeurs,
- peuvent accepter un nombre indéterminé d'arguments (non nommés au départ)
- permettent de nommer les arguments assignés ou de fournir des arguments complémentaires nommés. [2]

1. Cette valeur spéciale est appelée `null` dans de nombreux autres langages.
2. Cette dernière possibilité ne sera pas étudiée.

Une fonction peut s'exprimer, avec au choix, les arguments suivants :

```
def ma_fonction( argument, argument_optionel = -1, *args, **kwargs ):
```

- Si on ne précise rien, l'argument est un argument obligatoire : oublier sa valeur lors de l'appel de la fonction provoquera une erreur.
- Un argument optionel est un argument avec une valeur par défaut : si l'on ne précise pas sa valeur lors de l'appel, c'est la valeur par défaut qui sera utilisée.
- **args* est une liste d'arguments non nommés. Ils sont ajoutés en en fin d'appel de fonction. Un exemple typique est la définition d'une fonction *somme* acceptant un nombre illimité de termes.

Exemple de valeurs par défaut aux arguments d'une fonction :

norme.py

```
1  # -*- coding: utf-8 -*-
2
3  def norme(x, y=0, z=0):
4      return (x**2+y**2+z**2)**0.5
```

Ainsi, des appels à `norme(0,4,3)`, à `norme(4,3)`, à `norme(-5)`, à `norme(4,z=3)`, ou à `norme(0,z=4,y=3)` renvoient 5.0.
Un appel à `norme(y=4, z=3)` provoque une erreur : `x` n'a pas de valeur.

⇒ **Activité 6.33**

Expliquez les résultats du programme suivant :

foncarg.py

```
1  #-*- coding: utf-8 -*-
2
3  def fonc_arg1(a=1,b=1,c=1):
4      resultat=a+b+c
5      return resultat
6
7  print(fonc_arg1(3,5,7))
8  print(fonc_arg1(3,6))
9  print(fonc_arg1())
10
11 def fonc_arg2(a,b,c):
12     resultat=a+b+c
13     return resultat
14
15 print(fonc_arg2(3,5,7))
16 print(fonc_arg2(3,6))
```

Résultat :

```
15
10
3
15
Traceback (most recent call last):
  File "foncarg.py", line 17, in <module>
    print(fonc_arg2(3,6))
TypeError: fonc_arg2() missing 1 required positional argument: 'c'
```

6.6.8 Fonction avec nombre indéterminé d'arguments

Si cela peu paraître inutile, l'opportunité d'ajouter un nombre illimité de paramètres en fin de fonction peu s'avérer terriblement utile pour des fonctions de traitement.
Pour les puristes, il est certes possible de concevoir sa fonction pour imposer un argument de type liste mais cela peut alourdir inutilement l'utilisation de la syntaxe.

Voici l'exemple d'une fonction capable de sommer un nombre illimité d'arguments :

somme_inf.py

```
1  #-*- coding: utf-8 -*-
2
3  def sommeinf(*liste_arguments):
4      resultat=0
5      for terme in liste_arguments:
6          resultat+=terme
7      return resultat
8
9  print("somme = ",sommeinf(10,5,3,98,65,34))
```

6.7 Listes en compréhension

Les listes en compréhension permettent de créer des listes de façon concise et très élégante.

6.7.1 Copie de liste

La variable *el* (comme *élément*) va parcourir la liste *maliste* :

```
>>> maliste=[1,2,3,4,5]
>>> [el for el in maliste]
[1, 2, 3, 4, 5]
```

6.7.2 Filtrage par test

En utilisant l'instruction `if`, qui teste si la condition est vraie, on peut filtrer les éléments de la liste :

```
>>> [el for el in maliste if el>3]
[4, 5]
```

6.7.3 Filtrage par test de fonction

Les tests peuvent être sous forme de fonctions :

```
>>> def pair(nombre):
...     return nombre % 2 == 0
...
>>> [el for el in maliste if pair(el)]
[2, 4]
```

6.7.4 Application d'une fonction

On peut appliquer une fonction (définie préalablement) aux éléments d'une liste :

```
>>> [el**2 for el in maliste if pair(el)]
[4, 16]
```

⇒ **Activité 6. 34**

1. Écrivez une seule ligne de 43 caractères, contenant l'instruction `if`, et qui affiche une liste des carrés des nombres divisibles par 5 compris entre 0 et 100 inclus. (L'instruction `print()` et les espaces sont comptés dans les 43 caractères).
2. Vérifiez à l'aide d'une autre ligne que la précédente compte bien 45 caractères.

Le programme :

6.8 Fonctions astucieuses

Certaines fonctions comme `zip`, `map` et `lambda` sont illustrées dans les programmes suivants :

complement1.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 30 08:29:19 2017

@author: beutier.m
"""

# On peut utiliser la fonction enumerate qui retourne
# chaque élément et sa position dans l'ensemble :

ma_liste = ["zéro", 1, "deux", 3, 4, "Five"]
for i,el in enumerate(ma_liste):
    print(i, el)
```

```
14
15 # Pour faire la somme de deux listes terme à terme, on peut écrire :
16 liste_1 = [4, 5, 6]
17 liste_2 = [3, 10, 11]
18 s = 0
19 for i in range(0,len(liste_1)) :
20     s = s + liste_1[i] + liste_2[i]
21 print(s)
22 # Ou utiliser la fonction zip :
23 liste_1 = [4, 5, 6]
24 liste_2 = [3, 10, 11]
25 s = 0
26 for el1,el2 in zip(liste_1,liste_2) :
27     s = s + el1 + el2
28 print(s)
29
30 # Il est possible d'éviter une fonction pour éviter d'écrire
31 # une boucle avec la fonction map.
32 # Elle applique une fonction à chaque élément d'un ensemble.
33 def fonction (x) :
34     return x ** 2
35 liste_3 = [3, 7, 9]
36 liste_4 = map (fonction, liste_3)
37 print (liste_4)
38 print (list(liste_4))
```

complement2.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue May 30 08:29:19 2017
4
5  @author: beutier.m
6  """
7
8  # Le mot-clé lambda permet de définir des fonctions au sein d'une
                                          expression.
9  def fonction (x):
10     return x**3
11 liste_1 = [3, 7, 9]
12 liste_2 = map (fonction, liste_1)
13 print (list(liste_2))
14 # On peut l'écrire :
15 liste_1 = [3, 7, 9]
16 liste_2 = map (lambda x : x**3, liste_1)
17 print (list(liste_2))
18
19 # Et si on veut ajouter un paramètre à la fonction lambda :
20 liste_1 = [3, 7, 9]
21 k = 3
22 liste_2 = map (lambda x, y=k : x**k, liste_1)
23 print (list(liste_2))
24
25 # Fonction eval()
26 a = 39
27 b = 2
28 for op in ["+","-","*","/","//","%","|","&","or","and","==","<",">","
                                          <<",">>","is"]:
```

```
29      print(str(a)+" "+op+" "+str(b),"->", eval(str(a)+" "+op+" "+str(b
                                                  )))
30
31  print("39 : \t\t",format(a,"b").rjust(8,"0"))
32  print("2 : \t\t",format(b,"b").rjust(8,"0"))
33  print("39 + 2 : \t",format(41,"b").rjust(8,"0"))
34  print("39 - 2 : \t",format(37,"b").rjust(8,"0"))
35  print("39 or 2 : \t",format(39,"b").rjust(8,"0"))
36  print("39 and 2 : \t",format(2,"b").rjust(8,"0"))
37  print("39 < 2 : \t",format(156,"b").rjust(8,"0"), " : décalage 2 bits
                                              vers la gauche")
38  print("39 >> 2 : \t",format(9,"b").rjust(8,"0"), " : décalage de 2
                                             bits vers la droite")
```

À vous d'en faire bon usage, même si elle ne sont pas au programme...

6.9 Gestion des erreurs

Parfois, une erreur est générée par *Python* et cela se termine par un message d'insultes ainsi que l'arrêt du programme.

Pour éviter cela, il y a quatre mots-clés vont servir à gérer les erreurs.

`try` permet "de tester" un bout de code. Si une erreur est rencontrée, on cesse d'interpréter le code et on passe aux `except`, qui permettent d'agir en fonction de l'erreur qui s'est produite. C'est pourquoi `try` doit toujours être suivi d'au moins un `except` ou d'un `finally`.

Exemple avec le code suivant :

```
try:
      reponse = int(input('Entrez un nombre entier : '))
except:
      print("Une erreur est survenue")
else:
      print("Le nombre est ",reponse)
```

- Si un nombre entier est saisi, il est ensuite affiché,
- Si un flottant ou une chaîne ou autre est entré, le message "Une erreur est survenue" est affichée.

Différents types d'erreurs existent. On peut citer :

- `ZeroDivisionError`
- `NameError`
- `TypeError`
- `ValueError`
- `IOError`
- `IndexError`
- `RuntimeError`
- `ImportError`
- `TabError`

En règle générale, l'instruction `try` fonctionne ainsi :

- D'abord, la clause d'essai (clause `try` : les instructions entre les mots-clés `try` et `except`) est exécutée,

- S'il ne se produit pas d'exception, la clause d'exception (clause `except`) est ignorée, et l'exécution du `try` est alors terminée,
- Si une exception se produit à un moment de l'exécution de la clause d'essai, le reste de la clause `try` est ignoré. Ensuite, si son type correspond à l'exception donnée après le mot-clé `except`, la clause `except` est exécutée, puis l'exécution reprend après l'instruction `try`,
- Si une exception se produit qui ne correspond pas à l'exception donnée dans la clause `except`, elle est renvoyée aux instructions `try` extérieures ; s'il n'y a pas de prise en charge prévue, il s'agit alors d'une exception non gérée et l'exécution est arrêtée avec un message d'insultes !
- `else` permet d'exécuter la suite du programme dans le cas où aucune exception ne survient,
- `finally` applique toujours le bloc, qu'une exception soit levée ou non.

Une instruction `try` peut avoir plusieurs clauses d'exception, de façon à définir des gestionnaires d'exception différents pour des exceptions différentes.

Ainsi, il est possible d'avoir :

```
reponse = input('Entrez un nombre entier : ')

try:
      inverse = 1/float(reponse)
except ValueError:
      print(reponse,"n'est pas un nombre !")
except ZeroDivisionError:
      print("Division par zéro impossible !")
else:
      print("Son inverse est ",inverse)
```

On peut également indiquer plusieurs erreurs entre parenthèses :

```
reponse = input('Entrez un nombre entier : ')

try:
      inverse = 1/float(reponse)
except (ValueError,ZeroDivisionError):
      print("Un problème est survenu")
else:
      print("Son inverse est ",inverse)
```

Il est également possible de nommer et d'utiliser une erreur.

Un exemple pratique en utilisant `except...as` dans le programme ci-dessous :

try_except.py

```
1  # -*- coding: utf-8 -*-
2
3  from math import sin
4  for x in range(-3, 4):
5      try:
6          print("\nx = {:.1f} donne\
7           sin(x)/x = {:.3f}".format(x,sin(x)/x))
8      except ZeroDivisionError as erreur: # exception
9          print("\nx = 0.0 donne\
10          sin(x)/x = 1.000") # exception pour x = 0
11         print("Cette erreur a été évitée : ",erreur)
12
13 for x in range(-3, 4):
14     print("\nx = {:.1f} donne\
15     sin(x)/x = {:.3f}".format(x,sin(x)/x))
```

et le résultat :

```
x = -3.0 donne          sin(x)/x = 0.047

x = -2.0 donne          sin(x)/x = 0.455

x = -1.0 donne          sin(x)/x = 0.841

x = 0.0 donne          sin(x)/x = 1.000
Cette erreur a été évitée :  float division by zero

x = 1.0 donne          sin(x)/x = 0.841

x = 2.0 donne          sin(x)/x = 0.455

x = 3.0 donne          sin(x)/x = 0.047
```

Dans certains cas, on peut être amené à créer des exceptions personnalisées. L'instruction `raise` peut être utilisée à ce propos. On peut également utiliser l'instruction `assert` amis ce ne sera pas détaillé ici.

7

Algorithmes et programmes importants

7.1 Recherche dans une liste

⇒ **Activité 7.35**

L'algorithme suivant recherche un élément el dans une liste t.

Recherche dans un tableau

```
1   VARIABLES
2   el
3   i : int
4   t[1..n] : tableau
5   ENTRÉES
6   LIRE el.
7   SORTIES
8   AFFICHER si l'élément est présent ou non.
9   DEBUT_ALGORITHME
10      POUR i ALLANT_DE 1 A taille(t)
11         DEBUT_POUR
12         SI (el = t[i]) ALORS
13            DEBUT_SI
14            AFFICHER ("L'élément recherché a été découvert")
15            FIN_SI
16         SINON
17            DEBUT_SINON
18            AFFICHER ("L'élément recherché n'est pas présent")
19            FIN_SINON
20         FIN_POUR
21  FIN_ALGORITHME
```

Écrivez en *python* la fonction correspondante.

On obtient par exemple le programme *recherche-liste.py* :

7.2 Maximum et minimum d'une liste de valeurs

⇒ **Activité 7. 36**

Complétez l'algorithme suivant qui recherche le maximum et le minimum d'une liste t puis écrire le programme *python* correspondant. **On créera des fonctions** en *python*.

Mini et maxi d'une liste

```
1   VARIABLES
2   min, max : float
3   i : int
4   t[1..n] : tableau de nombres
5   ENTRÉES
6   LIRE t.
7   SORTIES
8   AFFICHER le maximum et le minimum
9   DEBUT_ALGORITHME
10      INITIALISATION
11      .........
12      .........
13      POUR i ALLANT_DE ... A ...
14         DEBUT_POUR
15         SI (t[i] < min) ALORS
16            DEBUT_SI
17            .........
18            FIN_SI
19         SINON
20            DEBUT_SINON
21            SI (t[i] > max) ALORS
22               DEBUT_SI
23               .........
24               FIN_SI
25            FIN_SINON
26         FIN_POUR
27      AFFICHER ("Le min est: ", min)
28      AFFICHER ("Le max est: ", max)
29  FIN_ALGORITHME
```

On peut aussi créer une fonction qui retourne 2 arguments (minimum et maximum) sous forme de tuple :

minetmax.py

```
1  # -*- coding: utf-8 -*-
2
3  tab=[12,24,98,2,-4,78.567,100]
4
5  def min_et_max(t):
6      mini,maxi = t[0],t[0]
7      for i in range(len(t)):
8          if maxi < t[i]:
9              maxi = t[i]
```

```
10          elif mini > t[i]:
11              mini = t[i]
12      return(mini,maxi)
13
14  print(min_et_max(tab))
15  # c'est un tuple
```

7.3 Variance d'une liste de valeurs

⇒ **Activité 7.37**

Soit une liste $t = [t[1],...,t[n]]$. La variance de t est définie par la relation :

$$var = \frac{1}{n}\sum_{i=1}^{n}(t[i] - t_m)^2$$

avec t_m moyenne de t.

L'algorithme suivant recherche la moyenne et la variance d'une liste t de nombres.

Moyenne et variance d'une liste

```
1   VARIABLES
2   moy, var, somme, sommecar : float
3   i : int
4   t[1..n] : tableau de nombres
5   ENTRÉES
6   LIRE t.
7   SORTIES
8   AFFICHER la moyenne et la variance
9   DEBUT_ALGORITHME
10      INITIALISATION
11      somme = 0
12      taille = 0
13      sommecar = 0
14      POUR i ALLANT_DE 1 A n
15          DEBUT_POUR
16          somme ← somme + t[i]
17          taille ← taille + 1
18          FIN_POUR
19      moy ← somme/taille
20      AFFICHER ("La moyenne est: ", moy)
21      POUR i ALLANT_DE 1 A n
22          DEBUT_POUR
23          sommecar ← sommecar + (t[i] − moy)^2
24          taille ← taille + 1
25          FIN_POUR
26      var ← sommecar/taille
27      AFFICHER ("La variance est: ", var)
28  FIN_ALGORITHME
```

Écrivez les fonctions correspondantes en *python*.
On obtient par exemple le programme *moy-var.py* :

7.4 Recherche par dichotomie dans un tableau trié

⇒ **Activité 7.38**

Complétez l'algorithme suivant puis écrivez le programme *python* correspondant permettant de trouver un élément dans un tableau trié.

Recherche dans un tableau

```
1   VARIABLES
2   a, b, c, elem, i : int
3   tab[1..n] : tableau trié de nombres
4   ENTRÉES
5   LIRE elem et tab.
6   SORTIES
7   AFFICHER si elem est dans tab
8   DEBUT_ALGORITHME
9      INITIALISATION
10      a ← 1
11      b ← taille(tab)
12      TANT_QUE ..... FAIRE
13         DEBUT_TANT_QUE
14         .....
15         SI ..... ALORS
16            DEBUT_SI
17            AFFICHER (elem, 'est trouvé')
18            FIN_SI
19         SINON
20            DEBUT_SINON
21            SI ..... ALORS
22               DEBUT_SI
23               .....
24               FIN_SI
25            FIN_SINON
26         SINON
27            DEBUT_SINON
28            .....
29            FIN_SINON
30         FIN_TANT_QUE
31      AFFICHER (elem, "non trouvé")
32   FIN_ALGORITHME
```

En *python*, on obtient par exemple le programme *recherche-tab.py* :

7.5 Recherche par dichotomie du zéro d'une fonction

⇒ **Activité 7.39**

Soit une fonction f continue et monotone, $[a, b]$ un intervalle de recherche, p la précision, telle que par exemple, $p = 10^{-prec}$.
On souhaite rechercher le zéro de cette fonction : la recherche s'arrêtera lorsque $b - a < p$.

L'algorithme peut s'écrire par exemple comme ceci :

Zéro d'une fonction par dichotomie

```
1   VARIABLES
2   prec : int
3   p, a, b, c : float
4   DEBUT_ALGORITHME
5      LIRE p
6      p ← 10^-prec
7      LIRE a
8      LIRE b
9      SI (f(a) * f(b) > 0) ALORS
10        DEBUT_SI
11        AFFICHER ("Pas de solution")
12        FIN_SI
13     SINON
14        DEBUT_SINON
15        TANT_QUE (b - a > p) FAIRE
16           DEBUT_TANT_QUE
17           c ← (a + b)/2
18           SI (f(a) * f(c) > 0) ALORS
19              DEBUT_SI
20              a ← c
21              FIN_SI
22           SINON
23              DEBUT_SINON
24              b ← c
25              FIN_SINON
26           FIN_TANT_QUE
27        FIN_SINON
28     AFFICHER a
29  FIN_ALGORITHME
```

Écrivez le programme *python* correspondant permettant de trouver le zéro de cette fonction dans l'intervalle considéré.
Vous créerez une fonction et obtiendrez par exemple le programme *zero-dicho.py* :

7.6 Intégrales : Méthode des rectangles et des trapèzes

L'intégration numérique est un outil indispensable en physique numérique.
Les méthodes numériques d'intégration d'une fonction sont nombreuses et les techniques très diverses. Des très simples, comme la méthode des rectangles aux très complexes comme certaines variétés de la méthode de Monte-Carlo. Nous n'aborderons ici que des méthodes simples voire simplistes.

7.6.1 Méthode des rectangles

Considérons une fonction continue $f(x)$ sur un intervalle $[a, b]$. Pour un physicien, intégrer une fonction signifie calculer l'aire ou la surface sous la courbe de la fonction $f(x)$ entre a et b.
La première méthode qui vient à l'esprit, c'est de découper l'aire entre la courbe $f(x)$, l'axe des x et les droites $x = a$ et $x = b$, en une multitude de petits rectangles de largeur faible h, et de hauteur $f(h)$. L'aire sous la courbe est obtenue en sommant tous ces petits rectangles.

Nous avons le choix entre trois techniques :

- faire coïncider le sommet haut gauche du rectangle avec la courbe : c'est la méthode des rectangles à gauche,

- faire coïncider le sommet haut droit du rectangle avec la courbe : c'est la méthode des rectangles à droite,
- faire coïncider le milieu du coté haut du rectangle avec la courbe : c'est la méthode du point milieu.

Nous choisirons la dernière, la plus précise.

Posons $h = \dfrac{b-a}{n}$, où n est le nombre de rectangles avec lesquels nous allons paver l'aire à calculer. Évidemment, plus n sera grand, et plus la précision du calcul sera grande (du moins en première approche). En voici une illustration :

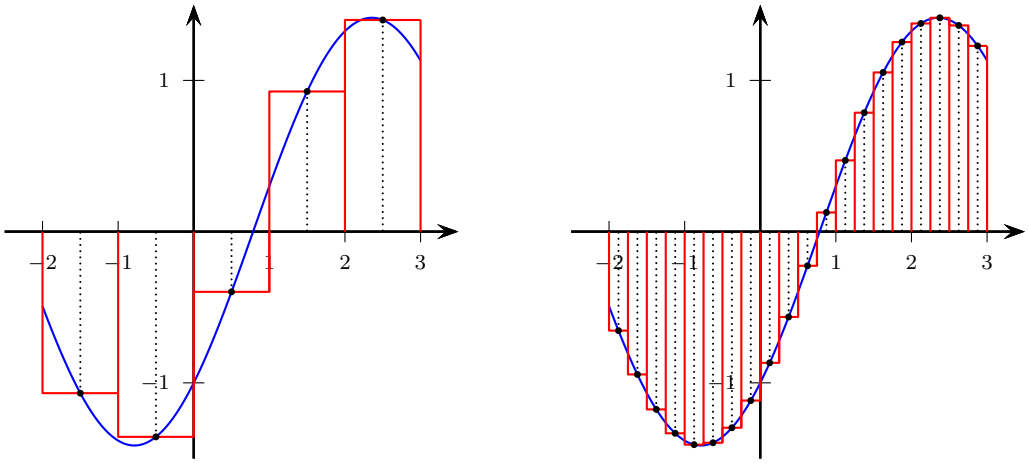


FIGURE 7.1 – Méthode des rectangles

Un rapide calcul nous montre que dans ce cas, l'intégrale, notée I, vaut :

$$I = \frac{b-a}{n} \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)$$

⇒ **Activité 7.40**

Soit l'algorithme suivant qui permet d'approcher une intégrale par la méthode des rectangles.
Écrivez le programme *python* correspondant à l'aide d'une fonction.

Intégrale, Méthodes des rectangles

```
1   DEBUT_ALGORITHME
2     VARIABLES
3     a, b, integrale : float
4     i, n : int
5     LIRE a
6     LIRE b
7     LIRE n
8     INITIALISATION
9     integrale ← 0
10     pas ← (b − a)/n
11     POUR i ALLANT_DE 0 A n − 1
12       DEBUT_POUR
13       integrale ← integrale + pas ∗ f(a + (i + 1/2) ∗ pas)
14       FIN_POUR
15     AFFICHER integrale
16  FIN_ALGORITHME
```

En *python*, on obtient par exemple le programme suivant *rect1.py* :

7.6.2 Méthode des trapèzes

La méthode des trapèzes est voisine de celle des rectangles. On utilise non plus des rectangles pour paver l'aire, mais des trapèzes. Ainsi, la partie du pavé qui jouxte la courbe est plus proche.
Comme plus haut, l'intervalle $[a, b]$ est partagée en n petits trapèzes de largeur $h = \dfrac{b-a}{n} = pas$.
L'aire de chaque petit trapèze est :

$$\begin{aligned} A_i &= \frac{h}{2}\left[f(x_i) + f(x_{i+1})\right] \\ &= \frac{h}{2}\left[f(a+i\,h) + f(a+(i+1)\,h)\right] \end{aligned}$$

On obtient l'aire recherchée en sommant l'aire de tous les trapèzes entre a et b, ce qui donne :

$$I = \frac{b-a}{2n}\left[f(a) + f(b) + 2\sum_{i=0}^{n-1} f(x_i)\right]$$

7.7 Recherche de la position d'un mot dans une chaîne

⇒ **Activité 7. 41**

L'algorithme suivant permet de trouver la position d'un mot dans une chaîne de caractères.
Traduisez cet algorithme en langage *python*.

Mot dans une chaîne

```
1   VARIABLES
2   taillemot, taillechaine, i, k : int
3   mot, chaine : string
4   LIRE mot
5   LIRE chaine
6   DEBUT_ALGORITHME
7      INITIALISATION
8      taillemot ← taille(mot)
9      taillechaine ← taille(chaine)
10      POUR k ALLANT_DE 1 A taillechaine − taillemot
11         DEBUT_POUR
12         i ← 0
13         TANT_QUE (i < taillemot) FAIRE
14            DEBUT_TANT_QUE
15            SI (mot[i]! = chaine[k + i]) ALORS
16               DEBUT_SI
17               i ← taillemot
18               FIN_SI
19            SINON
20               DEBUT_SINON
21               SI (i == taillemot − 1) ALORS
22                  DEBUT_SI
23                  AFFICHER (k)
24                  FIN_SI
25               SINON
26                  DEBUT_SINON
27                  i ← i + 1
28                  FIN_SINON
29               FIN_SINON
30            FIN_TANT_QUE
31         FIN_POUR
32      AFFICHER ("Le mot n'est pas présent")
33   FIN_ALGORITHME
```

En *python*, on peut écrire le programme *recherche-mot.py* :

8

Preuve d'un algorithme

8.1 Compétences exigées

Objectifs

Les capacités évaluées dans cette partie de la formation sont :

- justifier qu'une itération (ou boucle) produit l'effet attendu au moyen d'un invariant,
- démontrer qu'une boucle se termine effectivement.

8.2 Position du problème

Les itérations permettent d'écrire des programmes qui exécutent plusieurs fois les mêmes instructions. Ce sont les seules instructions qui nous permettent d'écrire un programme dont le temps d'exécution est arbitrairement long, ou même infini. Par exemple, l'algorithme suivant ne s'arrête jamais :

boucle infinie 1

```
1  TANT_QUE 1 = 1 FAIRE
2     DEBUT_TANT_QUE
3     ...
4     FIN_TANT_QUE
```

Un autre exemple :

boucle infinie 2

```
1  INITIALISATION
2  n ← 0
3  TANT_QUE n ⩾ 0 FAIRE
4     DEBUT_TANT_QUE
5     n ← n + 1
6     FIN_TANT_QUE
```

Avec les boucles, il y a deux questions qui surgissent :

- La **terminaison** : la boucle se termine-t-elle ? Sous quelles conditions ?
- La **validité** : que calcule la boucle en question ?

Pour les boucles `POUR...DE...A`, le problème de terminaison ne se pose pas. Il n'en est pas de même pour les boucles `TANT QUE`.

8.3 Terminaison

8.3.1 Méthode

Pour prouver qu'un algorithme faisant intervenir une boucle conditionnelle se termine, on cherche à identifier une expression à valeurs entières positives strictement décroissante et minorée : mathématiquement, cela se traduit par :

$$\exists m \in \mathbb{R} \text{ tq } \forall n \in \mathbb{N}, m \leqslant u_n$$

Cette expression est appelée **variant de boucle**.

8.3.2 Exemple

On peut par exemple considérer l'algorithme suivant qui effectue la division euclidienne d'un entier a par un autre entier b :

division euclidienne

```
1   VARIABLES
2   a, b : entiers
3   SORTIES
4   AFFICHER le quotient q et le reste r
5   DEBUT_ALGORITHME
6      INITIALISATION
7      q ← 0
8      r ← a
9      TANT_QUE r ⩾ b FAIRE
10        DEBUT_TANT_QUE
11        r ← r − b
12        q ← q + 1
13        FIN_TANT_QUE
14     AFFICHER quotient q et reste r
15  FIN_ALGORITHME
```

Cet algorithme permet de calculer le quotient q et le reste r de la division de a par b.

Principe

À chaque itération de la boucle, r est minoré par b et décroît strictement puisque b est strictement positif. Ceci assure la terminaison de la boucle.

En sortie de boucle, on sait que $r < b$ mais comme r est positif, r est en réalité compris dans l'intervalle $[0, b-1]$

8.4 Les invariants de boucle

Il est indispensable qu'un algorithme se termine mais on souhaite également qu'il effectue la tâche voulue !

C'est ce qu'on vérifie au moyen d'un **invariant de boucle**.

Remarque

Lorsqu'il est indiqué dans la structure de l'algorithme, l'invariant de boucle est en principe noté entre accolades, en tant que commentaire.

8.4.1 Définition

Invariant de boucle

On appelle invariant de boucle une proposition vérifiant les conditions suivantes :

- La proposition est vraie avant l'entrée dans la boucle.
- Si cette proposition est vraie au début d'une itération, elle reste vraie à la fin de l'itération, donc au début de l'itération suivante.

Dans ce cas, cette proposition sera vraie à la sortie de la boucle.

Il s'agit en fait d'un raisonnement par récurrence.

8.4.2 Exemple

Reprenons l'exemple de la division euclidienne de a par b.

- Initiation : avant la boucle, $q = 0$ et $r = a$. On a donc $a = b \times q + r$.
- Hérédité : supposons que $a = b \times q + r$ au début d'une itération. Notons q' et r' les valeurs de q et r à la fin de l'itération.
  On a $q' = q + 1$ et $r' = r - b$ de sorte que $b \times q' + r' = b \times (q + 1) + (r - b) = b \times q + r = a$. En fin d'itération, on a alors $a = b \times q + r$ avec $r \in [0, b - 1]$ (condition de sortie), ce qui assure que q et r sont bien le quotient et le reste de la division euclidienne de a par b.

$\{a = b \times q + r\}$ constitue donc un invariant de boucle.

On peut le montrer en remplissant le tableau suivant, en prenant par exemple $a = 17$ et $b = 4$:

Itération	q	r	$b \times q + r$
Avant la boucle			
première itération			
deuxième itération			
troisième itération			
quatrième itération			

8.4.3 Implémentation en *Python*

⇒ **Activité 8. 42**

1. Proposez une fonction permettant de calculer le résultat de la division euclidienne de a par b.
2. Introduisez dans la fonction un test permettant de savoir si l'invariant de boucle est vérifié à chaque itération.

Le résultat :

```
Division euclidienne de 17 par 4 :
Invariant vérifié
Invariant vérifié
Invariant vérifié
Invariant vérifié
(4, 1)
```

8.5 Applications

8.5.1 Algorithme d'Euclide

8.5.1.1 Algorithme

Soit l'algorithme suivant qui calcule le plus grand commun diviseur (pgcd) n de deux entiers a et b :

algorithme d'Euclide

```
1   VARIABLES
2   a, b : entiers
3   n, p : entiers
4   SORTIES
5   AFFICHER le pgcd n de a et b
6   DEBUT_ALGORITHME
7      INITIALISATION
8      n ← a
9      p ← b
10     TANT_QUE p ≠ 0 FAIRE
11        DEBUT_TANT_QUE
12        n, p ← p, n%p
13        # n%p est le reste de n/p
14        FIN_TANT_QUE
15     AFFICHER le pgcd n
16  FIN_ALGORITHME
```

8.5.1.2 Terminaison

⇒ **Activité 8. 43**

Vérifiez la terminaison de cet algorithme.

8.5.1.3 Invariant de boucle

⇒ **Activité 8. 44**

Vérifiez que $\{pgcd(a,b) = pgcd(n,p)\}$ constitue un invariant de boucle.

8.5.1.4 Implémentation en *Python*

⇒ **Activité 8. 45**

1. Proposez une fonction permettant de calculer le pgcd de a et b de façon itérative.
2. Proposez une fonction permettant de calculer le pgcd de a et b de façon récursive.
3. Introduisez dans la deuxième fonction un test permettant de savoir si l'invariant de boucle est vérifié à chaque itération. Vous pourrez utiliser la première fonction pour vérifier cet invariant dans la deuxième fonction.

Le résultat :

```
pgcd de 56 et 42 :
14
Invariant vérifié
Invariant vérifié
14
```

8.5.2 Puissance

⇒ **Activité 8. 46**

Soit n un entier strictement positif et a un réel. Considérons l'algorithme suivant qui calcule $s = a^n$:

puissance

```
1   VARIABLES
2   a, r, s : flottants
3   n, N : entiers
4   SORTIES
5   AFFICHER a^n
6   DEBUT_ALGORITHME
7      INITIALISATION
8      s ← a
9      N ← n
10      r ← 1
11     TANT_QUE N > 0 FAIRE
12        DEBUT_TANT_QUE
13        SI N%2 = 0 # (N pair) ALORS
14           DEBUT_SI
15           s ← s × s
16           N ← N/2
17           FIN_SI
18        SINON
19           DEBUT_SINON
20           r ← r × s
21           N ← N − 1
22           FIN_SINON
23        FIN_TANT_QUE
24     AFFICHER s
25  FIN_ALGORITHME
```

Montrez que l'algorithme se termine et vérifiez que l'invariant de boucle est $\{s^N \times r = a^n\}$.

8.5.3 Factorielle

⇒ **Activité 8. 47**

Considérons l'algorithme suivant qui calcule la factorielle res d'un entier n :

factorielle

```
1   VARIABLES
2   n, i, res : entiers
3   SORTIES
4   AFFICHER res
5   DEBUT_ALGORITHME
6      INITIALISATION
7      i ← 0
8      res ← 1
9      TANT_QUE i < n FAIRE
10        DEBUT_TANT_QUE
11        i ← i + 1
12        res ← res × i
13        FIN_TANT_QUE
14     AFFICHER res
15  FIN_ALGORITHME
```

Montrez que l'algorithme se termine et vérifiez que $\{res = i!\}$ est un invariant de boucle.

9

Complexité d'un algorithme

9.1 Compétences exigées

Objectifs

La capacité évaluée dans cette partie de la formation est :

- s'interroger sur l'efficacité algorithmique temporelle d'un algorithme.

9.2 Notion de complexité

Il existe souvent plusieurs façons de programmer un algorithme. Si le nombre d'opérations à effectuer est peu important et les données d'entrée de l'algorithme sont de faibles tailles, le choix de la solution importe peu. En revanche, lorsque le nombre d'opérations et la taille des données d'entrée deviennent importants, deux paramètres deviennent déterminants : le temps d'exécution et l'occupation mémoire.
On n'exige donc pas seulement d'un algorithme qu'il résolve un problème, on veut également qu'il soit efficace, c'est-à-dire :

- rapide (en termes de temps d'exécution),
- peu gourmand en ressources (espace de stockage, mémoire utilisée).

On a alors besoin d'outils qui nous permettront d'évaluer la qualité théorique des algorithmes proposés. Le coût de la mémoire étant aujourd'hui relativement faible, on cherche en général à améliorer la complexité en temps plutôt que la complexité en mémoire. Pour cette raison, on s'intéressera donc au temps d'exécution.

Complexité

La complexité d'un algorithme en temps donne le nombre d'opérations effectuées lors de l'exécution d'un programme.
On appelle C_i le coût en temps d'une opération i.

La complexité en mémoire donne le nombre d'emplacements mémoires occupés lors de l'exécution d'un programme.

La complexité permet de :

- classer les problèmes selon leur difficulté,
- classer les algorithmes selon leur efficacité,
- comparer les algorithmes correspondant à un problème donné sans avoir à les implémenter, c'est-à-dire à les traduire dans un langage particulier.

Remarque

Attention de ne pas confondre la complexité d'un algorithme et la complexité d'un problème : la complexité d'un problème correspond à la complexité de l'algorithme le plus efficace résolvant ce problème.

9.3 Calcul de la complexité

9.3.1 Les opérations

L'efficacité d'un algorithme ne se mesure pas en secondes, par exemple, car cela impliquerait justement de les implémenter mais de plus, ces mesures ne seraient pas significatives car dépendantes de la machine utilisée.

On utilise donc des unités de temps abstraites correspondant au nombre d'opérations effectuées. Chaque opération (addition, multiplication, comparaison, incrémentation, affectation, ...) consomme alors une unité de temps. Le nombre d'opérations est égal à la somme de toutes les opérations.

Exemple du calcul de *factorielle (n)* :

Complexité 1

```
1   VARIABLES
2   i, n, fact : int
3   SORTIES
4   AFFICHER factorielle
5   DEBUT_ALGORITHME
6      INITIALISATION
7      fact ← 1                          Initialisation : 1
8      POUR i ALLANT_DE 2 A n            Itérations : n − 1
9         DEBUT_POUR
10          fact ← fact ∗ i               multiplication + affectation : 2
11        FIN_POUR
12      AFFICHER fact                   affichage : 1
13  FIN_ALGORITHME
```

Il y a aussi des opérations cachées :

- i est initialisé à 2 en début de boucle : 1 opération,
- pour chaque boucle, i est incrémenté, affecté, puis comparé à n : 3 opérations.

Il y a donc au total :
$1 + 1 + 5 \times (n-1) + 1 = 5n - 2$ opérations.

9.3.2 Complexité dans les différents cas

Il est fréquent que la complexité en temps soit améliorée au prix d'une augmentation de la complexité en espace, et vice-versa. La complexité dépend notamment :

- de la puissance de la machine sur laquelle l'algorithme est exécuté,
- du langage et compilateur (ou interpréteur) utilisé pour coder l'algorithme,

- du style du programmeur.

On distingue la complexité dans le pire des cas, la complexité dans le meilleur des cas, et la complexité en moyenne. En effet, pour un même algorithme, suivant les données à manipuler, le résultat sera déterminé plus ou moins rapidement.

- La complexité au meilleur est le plus petit nombre d'opérations qu'aura à exécuter l'algorithme sur un jeu de données de taille fixée. C'est une borne inférieure de la complexité de l'algorithme sur un jeu de données de taille n.
- La complexité au pire est le plus grand nombre d'opérations qu'aura à exécuter l'algorithme sur un jeu de données de taille fixée. Comme il s'agit d'un maximum, l'algorithme finira donc toujours avant d'avoir effectué ce nombre maximum d'opérations.
  Cette complexité peut cependant ne pas refléter le comportement " usuel " de l'algorithme, le pire cas ne pouvant se produire que rarement.
- La complexité en moyenne est plus difficile à calculer. Il ne s'agit pas comme on pourrait le penser de la moyenne des complexités au mieux et au pire. Concrètement, on se donne la probabilité d'apparition de chacun des jeux de données.
  Cette complexité en moyenne reflète le comportement " général " de l'algorithme si les cas extrêmes sont rares ou si la complexité varie peu en fonction des données.
  La complexité en pratique sur un jeu de données particulier peut être nettement plus importante que la complexité en moyenne : dans ce cas la complexité en moyenne ne donnera pas une bonne indication du comportement de l'algorithme.

Généralement, on s'intéresse au cas le plus défavorable, à savoir, la complexité dans le pire des cas, notée $\mathcal{O}(f)$, car cela correspond à la performance de l'algorithme lorsqu'il prend le plus de temps : il ne prendra jamais plus de temps que ce qu'on a estimé.
La complexité en moyenne est également souvent utilisée.

9.3.3 Simplifications : Règles de calcul

Partons d'un algorithme effectuant $4n^3 + 5n^2 - 2n + 8$ opérations.
Les processeurs actuels effectuent plusieurs millions d'opérations par seconde : ainsi, qu'une affectation requière 2, ou bien 4 unités de temps, ne modifie pas fondamentalement les choses.
On remplace donc les constantes multiplicatives par des 1, ce qui donne : $n^3 + n^2 - 2n + 8$.
Un nombre constant d'instructions est négligeable par rapport à la croissance de la taille des données, ce qui permet d'annuler les constantes additives. Cela donne : $n^3 + n^2 - 2n$.
Pour de grandes valeurs de n, c'est bien sûr le terme de plus haut degré qui est prépondérant.
Il reste alors : n^3.

9.3.4 Classes de complexités

La notation $\mathcal{O}(f)$ décrit le comportement asymptotique d'une fonction, c'est à dire pour des grandes valeurs.
Les complexités d'algorithmes (dans le pire des cas, que l'on notera $\mathcal{O}$) les plus courantes sont :

Complexité	temps d'éxécution
$\mathcal{O}(1)$	temps constant
$\mathcal{O}(\log(n))$	logarithmique
$\mathcal{O}(n)$	linéaire
$\mathcal{O}(n * log(n))$	linéarithmique
$\mathcal{O}(n^p)$	polynomiales
$\mathcal{O}(p^n)$	exponentielle

TABLE 9.1 – Complexités d'algorithmes

Un algorithme logarithmique est considérablement plus efficace qu'un algorithme linéaire (lui-même plus efficace qu'un algorithme quadratique ou pire exponentiel). Cette différence devient fondamentale si la taille des données est importante : le résultat peut se compter en millisecondes pour une méthode et en heures pour une autre !

Avec 10^9 instructions par seconde, on aurait les temps d'exécution :

Complexité	temps d'exécution en fonction du nombre d'opérations					
n	5	10	15	20	100	1000
$\log n$	3.10^{-9} s	4.10^{-9} s	4.10^{-9} s	5.10^{-9} s	7.10^{-9} s	10^{-8} s
$2n$	10^{-8} s	2.10^{-8} s	3.10^{-8} s	4.10^{-8} s	2.10^{-7} s	2.10^{-6} s
$n \log n$	$1{,}2.10^{-8}$ s	3.10^{-8} s	6.10^{-8} s	10^{-7} s	7.10^{-7} s	10^{-5} s
n^2	$2{,}5.10^{-8}$ s	10^{-7} s	$2{,}3.10^{-7}$ s	4.10^{-7} s	10^{-5} s	10^{-3} s
n^5	3.10^{-6} s	10^{-4} s	$7{,}6.10^{-4}$ s	3.10^{-3} s	10 s	10^6 s 11 jours
2^n	$3{,}2.10^{-8}$ s	10^{-6} s	$3{,}3.10^{-5}$ s	10^{-3} s	$1{,}2.10^{21}$ s 4.10^{13} ans	10^{292} s 3.10^{284} ans
$n!$	$1{,}2.10^{-7}$ s	4.10^{-3} s	$1{,}4.10^{3}$ s 23 minutes	$2{,}4.10^{9}$ s 77 ans	10^{147} s 3.10^{141} ans	10^{500} s
n^n	3.10^{-6} s	10 s	$4{,}4.10^{8}$ s 13 ans	10^{17} s 3.10^{9} ans	10^{191} s 3.10^{183} ans	10^{300} s

TABLE 9.2 – Temps d'exécution d'algorithmes

9.4 Exemple : Exponentiation rapide

9.4.1 Méthode naïve

9.4.1.1 Algorithmique

Calculer a^n nécessite a priori $n-1$ multiplications selon l'algorithme suivant :

Exponentiation naïve

```
1   VARIABLES
2   a : float
3   n : int
4   ENTRÉES
5   un réel a et une puissance n
6   INITIALISATION
7   Resultat ← a
8   DEBUT_ALGORITHME
9      POUR k ALLANT_DE 2 A n
10        DEBUT_POUR
11        Resultat ← Resultat × a
12        FIN_POUR
13     AFFICHER Resultat
14  FIN_ALGORITHME
```

9.4.1.2 Avec *Python*

puiss-naive.py

```
1  # -*- coding: utf8 -*-
2
```

```
3  def puissance_naive(a,n):
4      res=a
5      for k in range(1,n):
6          res*=a
7      return res
8
9  # print(puissance_naive(2,100000))
10 from timeit import Timer
11 duree = Timer("puissance_naive(2,1000000)",\
12 "from __main__ import puissance_naive")
13 print("durée = %2.3f" % (duree.timeit(1)))
```

9.4.2 Algorithme de Hörner

9.4.2.1 Conventions

Dans la suite, nous confondrons polynôme et fonction polynôme.

9.4.2.2 Principe

Prenons l'exemple de $P(x) = 3x^5 - 2x^4 + 7x^3 + 2x^2 + 5x - 3$. Le calcul classique nécessite 5 additions et 15 multiplications.

On peut faire pas mal d'économies de calcul en suivant le schéma suivant :

$$\begin{aligned} P(x) &= \underbrace{a_n x^n + \cdots + a_2 x^2 + a_1 x}_{\text{on met } x \text{ en facteur}} + a_0 \\ &= \left(\underbrace{a_n x^{n-1} + \cdots + a_2 x}_{\text{on met } x \text{ en facteur}} + a_1 \right) x + a_0 \\ &= \cdots \\ &= (\cdots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})x + \cdots)\, x + a_0 \end{aligned}$$

Ici cela donne $P(x) = (((((3x) - 2)x + 7)x + 2)x + 5)x - 3$ c'est-à-dire 5 multiplications et 5 additions. En fait il y a au maximum 2 fois le degré de P opérations (voire moins avec les zéros).

9.4.2.3 L'algorithme

On peut essayer de faire mieux.

Voyons une première méthode qui utilise l'algorithme de HÖRNER étudié précédemment et la décomposition en base 2 de l'exposant.

Par exemple, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$. On peut donc associer à cette écriture un polynôme P tel que 11 soit égal à $P(2)$:

$$P(X) = 1 \times X^3 + 0 \times X^2 + 1 \times X^1 + 1 \times X^0 = ((X + 0)X + 1)X + 1$$

Donc

$$\begin{aligned} a^{P(2)} &= a^{1+2(1+2(2\times 1+0))} \\ &= a\cdot a^{2(1+2(2\times 1+0))} \\ &= a\cdot\left(a^{1+2(2\times 1+0)}\right)^2 = a^1\cdot\left(a^1\cdot\left(a^0\cdot\left(a^1\right)^2\right)^2\right)^2 \\ &= \left(\left(\left(a^1\right)^2 a^0\right)^2 a^1\right)^2 a^1 \end{aligned}$$

On en déduit l'algorithme :

Exponentiation « à la HÖRNER »

```
1   VARIABLES
2   a : float
3   n : int
4   ENTRÉES
5   un réel a et un entier n
6   INITIALISATION
7   C ← décomposition de n en base 2
8   res ← a
9   DEBUT_ALGORITHME
10      POUR j ALLANT_DE 1 A taille de C − 1
11         DEBUT_POUR
12         res ← res × res
13         SI le j-ème chiffre de la décomposition en base 2 est 1 ALORS
14            DEBUT_SI
15            res ← a × res
16            FIN_SI
17         AFFICHER res
18         FIN_POUR
19   FIN_ALGORITHME
```

Pour calculer a^{11} nous n'avons donc plus à effectuer que 6 multiplications ou plutôt 5 si on ne compte pas la multiplication par 1.

9.4.2.4 Avec Python

puiss-horner.py

```python
# -*- coding: utf8 -*-

def base2(n):
    r=n%2
    q=n//2
    R=[r]
    while q>0 :
        r=q%2
        q=q//2
        R=[r]+R
    return R

def puissance_horner(a,n):
    C=base2(n)
    res=a
    for j in range(1,len(C)):
        res*=res
        if C[j]==1: res*=a
    return(res)

# print(puissance_horner(2,100000))
```

```
22 import timeit
23 duree = timeit.Timer("puissance_horner(2,1000000)",\
24 "from __main__ import puissance_horner")
25 print("durée = %.3e" % duree.timeit(1))
```

9.4.3 Variante sans utiliser l'écriture en base 2

9.4.3.1 Algorithme

Voyons comment procéder sur notre exemple habituel :

$a^{11} = a \cdot a^{10} = a \cdot \left(a^5\right)^2 = a \cdot \left(a \cdot a^4\right)^2 = a \cdot \left(a \cdot \left(a^2\right)^2\right)^2$

Ainsi, on réduit l'exposant k selon sa parité :

- si k est pair, on écrit $a^k = \left(a^{\frac{k}{2}}\right)^2$;
- sinon, $a^k = a \cdot \left(a^{\frac{k-1}{2}}\right)^2$

Exponentiation rapide sans utiliser la base 2

```
1  VARIABLES
2  a : float
3  n : int
4  ENTRÉES
5  un réel a et un entier n
6  INITIALISATION
7  res ← 1
8  puissance ← n
9  temp ← a
10  DEBUT_ALGORITHME
11     TANT_QUE puissance non nulle FAIRE
12        DEBUT_TANT_QUE
13        SI puissance est impaire ALORS
14           DEBUT_SI
15           res ← temp × res
16           puissance ← puissance − 1
17           FIN_SI
18        puissance ← puissance/2
19        temp ← temp × temp
20        FIN_TANT_QUE
21  FIN_ALGORITHME
```

9.4.3.2 Avec *Python*

puiss-rapid.py

```
1  # -*- coding: utf8 -*-
2
3  def expo_rapid(a,n):
4      res=1
5      Puissance=n
6      temp=a
7      while Puissance>0:
8          if Puissance%2==1 :
9              res*=temp
10             Puissance+=-1
11         Puissance=Puissance/2
12         temp*=temp
```

```
13      return(res)
14
15  from timeit import Timer
16  duree = Timer("expo_rapid(2,1000000)",\
17  "from __main__ import expo_rapid")
18  print("durée = %.3e" % duree.timeit(1))
```

9.4.4 Comparaison des performances

Grâce à la méthode `Timer` du module `timeit`, nous pouvons comparer les temps de calcul des différents algorithmes pour calculer $2^{1\,000\,000}$:

```
from timeit import Timer
duree = Timer("puissance_naive(2,1000000)",\
"from __main__ import puissance_naive")
print("durée = %2.3f" % duree.timeit(1))
```

donne sur un ordinateur récent :

```
durée =  17.896
```

Avec *puiss_rapid* la durée d'exécution est de $4.540e-03$ secondes et avec *puiss_horner* $3.195e-03$ secondes !

En ajoutant un 0 à la puissance, la durée d'exécution de *puiss_naive* est extrêmement longue alors qu'avec *puiss_rapid* la durée d'exécution est de 0.0741 secondes et avec *puiss_horner* 0.03336 secondes !

Nous voyons ici que certains algorithmes sont plus efficaces que d'autres ...
Avec des algorithmes récursifs, la durée d'exécution peut même être encore diminuée : nous verrons cela plus tard ...

Remarque

Les écarts sont plus significatifs lorsque les nombres sont grands : si l'on teste avec des nombres petits, il est même probable que les durées d'exécutions se trouvent en ordre inverse.

10

Exercices python

Les programmes seront rendus sous le nom *(votre nom)-programme.py*.

► Exercice 10. 11 : Premiers nombres entiers : python

Écrivez un programme qui demande à l'utilisateur de saisir un nombre n puis affiche les n premiers nombres entiers.

► Exercice 10. 12 : Premiers entiers impairs : python

Écrivez un programme qui demande à l'utilisateur de saisir un nombre n puis affiche les n premiers entiers impairs.

► Exercice 10. 13 : Somme des premiers entiers pairs : python

Écrivez un programme qui demande à l'utilisateur de saisir un nombre n puis calcule la somme des n premiers entiers pairs en commençant par 2.

► Exercice 10. 14 : Intérêts : python

Écrivez un programme qui, à partir d'un montant à épargner et un taux d'intérêt annuel, calcule et affiche le montant augmenté des intérêts pour les n années à venir (vous afficherez le résultat avec 2 décimales).

► Exercice 10. 15 : Impôt sur les bénéfices : python

Écrivez un programme qui calcule l'impôt sur le bénéfice d'une société, le montant du bénéfice étant demandé à l'utilisateur, le montant de l'impôt étant de 20 % si le bénéfice est inférieur à 10000 €, de $2000 + 25$ % si le bénéfice est compris entre 10000 et 15000 € et de $3000 + 30$ % si le bénéfice est supérieur à 15000 €.

► Exercice 10. 16 : Fonction racine carrée

Écrivez un programme qui demande un flottant et qui calcule sa racine carrée avec 3 chiffres après la virgule s'il est positif ou nul.
Sinon affichez un message d'erreur.
On pourra utiliser :

```
from math import sqrt
```

► Exercice 10. 17 : Fonctions diverses

Écrivez une fonction `vols1` qui calcule directement le volume d'une sphère de rayon r fourni en argument.
Écrivez une fonction `cube` qui retourne le cube de son argument.
Écrivez une fonction `vols2` qui calcule le volume d'une sphère de rayon r fourni en argument et qui utilise la fonction cube.
Vous pourrez utiliser :

```
from math import pi
```

► Exercice 10. 18 : Suite 1

Considérons la suite (u_n) définie par :

$$\begin{cases} u_0 &= 1 \\ \forall\, n \in \mathbb{N}, \quad u_{n+1} &= \dfrac{u_n\left(6-u_n^2\right)}{4} \end{cases}$$

1. Calculez u_1 sans ordinateur.
2. Écrivez une fonction $u1\,(n)$ permettant de calculer u_n en fonction de n à l'aide d'une boucle `for`.
3. Écrivez une fonction $u2\,(n)$ permettant de calculer u_n en fonction de n à l'aide d'une boucle `while`.
4. Calculez $u1\,(1)$, $u1\,(1000)$, $u2\,(1)$ et $u2\,(1000)$.
5. Comparez le résultat obtenu avec :

```
>>> import math
>>> math.sqrt(2)
```

► Exercice 10. 19 : Suite 2

Considérons la suite (v_n) définie par :

$$\begin{cases} v_0 &= 1 \\ \forall\, n \in \mathbb{N}, \quad v_{n+1} &= -\dfrac{v_n}{(2\,n+1)\,(2\,n+2)} \end{cases}$$

1. (a) Calculez v_1 sans ordinateur.
   (b) Écrivez une fonction $v\,(n)$ permettant de calculer v_n en fonction de n.
2. Soit $s\,(n) = \sum_{i=0}^{i=n} v_i$.
   (a) Calculez s_1.
   (b) Écrivez une fonction $s(n)$ permettant de calculer s_n en fonction de n.
3. Calculez $s\,(1000)$.
4. Comparez le résultat obtenu avec :

```
>>> import math
>>> math.cos(1)
```

► Exercice 10. 20 : Suite 3

1. Écrivez une fonction qui calcule la somme des carrés de 1 à n : $somme1\,(n) = \sum_{k=1}^{n} k^2$.

   Remarque : on pourrait utiliser la fonction $somme2\,(n)$ qui effectue le même calcul en utilisant une (ou des) liste(s) ainsi que la fonction `sum` (cf. page 385).

suite3.py

```
1  def somme2 (n):
2      return sum([cpt**2 for cpt in range(1,n+1)])
```

2. Écrivez une fonction qui calcule le produit des carrés de 1 à n : $produit1\,(n) = \prod_{k=1}^{n} k^2$. Remarque : on pourrait utiliser la fonction $produit2\,(n)$ qui effectue le même calcul en utilisant une (ou des) liste(s) même si cela présente assez peu d'intérêt...

suite3.py

```
1  def produit2 (n):
2      prod = 1
3      L = [cpt for cpt in range(1,n+1)]
4      for el in L:
5          prod = prod * el**2
6      return prod
```

▶ Exercice 10.21 : Suite 4

Considérons la suite (u_n) définie par :

$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, \quad u_{n+1} = \sqrt{1+u_n} \end{cases}$$

1. Calculez u_1 et u_2 sans ordinateur.
2. Écrivez une fonction $u1\,(n)$ permettant de construire la liste u_0, u_1, ...,u_n en fonction de n. Vous pourrez utiliser :

```
from math import sqrt
```

3. Soit la fonction $u2\,(n)$:

suite4.py

```
1  def u2 (n):
2      return [sqrt(1+cpt) for cpt in range(0,n+1)]
```

Que fait cette fonction ? Cela répond-il à la question précdente ?

4. Modifiez la fonction $u1\,(n)$ afin qu'elle prenne en paramètre la valeur de u_0 de façon à pouvoir être modifiée par l'utilisateur.
5. Soit la fonction $u3\,(n)$:

suite4.py

```
1  def u3 (n,u_0):
2      ma_liste = [u_0]
3      for cpt in range(1,n+1):
4          ma_liste.append(sqrt(1+ma_liste[cpt-1]))
5      return ma_liste
```

Que fait cette fonction ?

▶ Exercice 10. 22 : Année bissextile : python

Écrivez un programme (si possible une fonction, par exemple `biss`) qui détermine si une année n est bissextile.
On rappelle que si n n'est pas divisible par 4, l'année n'est pas bissextile.
Si n est divisible par 4, l'année est bissextile sauf si n est divisible par 100 et pas par 400.
On pourra se reporter à l'algorithme page 29.
Dans un deuxième temps, on pourra écrire une fonction `compt_biss` qui compte les années bissextiles entre 2 années prises en argument.

▶ Exercice 10. 23 : Devinette 1 : python

Écrivez un programme dans lequel l'utilisateur doit deviner un nombre pair compris entre 10 et 100 généré par l'ordinateur.
Vous pourrez utiliser :

```
import random
# nombre aléatoire compris entre 10 et 100
n = random.randint(10, 100)
```

▶ Exercice 10. 24 : Devinette 2 : python

Écrivez un programme dans lequel l'ordinateur devine un nombre pair entre 0 et 100 choisi par l'utilisateur (version dichotomique).

▶ Exercice 10. 25 : pgcd

Le pgcd de deux entiers a et b peut être trouvé grâce à l'algorithme suivant :

Euclide

```
1   VARIABLES
2   a, b, r : int
3   DEBUT_ALGORITHME
4      LIRE a et b
5      TANT_QUE b ≠ 0 FAIRE
6         DEBUT_TANT_QUE
7         r ← reste(a,b) # ou a%b
8         a ← b
9         b ← r
10         FIN_TANT_QUE
11       AFFICHER a
12   FIN_ALGORITHME
```

Écrivez une fonction qui calcule le pgcd de deux entiers a et b.

▶ Exercice 10. 26 : ppcm

Écrivez une fonction qui calcule le ppcm de deux entiers a et b : le ppcm de a et b est donné par le quotient du produit de a et b et du pgcd de a et b.

▶ Exercice 10. 27 : Méthode des trapèzes

Écrivez les algorithme et fonction en python correspondant à la méthode des trapèzes abordée page 94.

▶ Exercice 10. 28 : Table de multiplication

1. Écrivez un programme qui interroge l'utilisateur sur une multiplication de deux nombres compris et choisis aléatoirement entre 1 et 10.
2. Modifiez le programme précédent de façon à ce que l'ordinateur affiche "Bravo" ou "Dommage" en fonction de la réponse de l'utilisateur.

3. Créez ensuite une boucle `for` afin que l'ordinateur fasse une série de 10 multiplications.

4. Comptez ensuite les bonnes réponses de façon à afficher en fin de programme :

   (a) "Félicitations : tant de bonnes réponses, tant de mauvaises sur tant."

   (b) "C'est moyen : tant de bonnes réponses, tant de mauvaises sur tant."

   (c) "Retournez en CE2 : tant de bonnes réponses, tant de mauvaises sur tant."

   Les seuils pourront par exemple être mis à 0,8 et 0,5.

5. Modifiez le programme précédent pour qu'il demande à l'utilisateur s'il veut refaire une autre série de multiplications. Si oui, le programme devra revenir au début par l'intermédiaire d'une boucle `while`. Si non, le programme s'arrêtera.

   ⚠ À chaque série, les compteurs intermédiaires de réponses devront être remis à 0.

6. Complétez le programme pour qu'il affiche en sortant le nombre total de bonnes réponses sur le total des questions (forcément un multiple de 10 pour ce dernier nombre).

7. Utilisez un chronomètre de façon à ce que la réponse soit comptée bonne si elle est donnée dans un laps de temps limité, 5 secondes, par exemple. Si la réponse est trop tardive, l'ordinateur affichera le temps de réponse.
   Complétez ensuite le programme de façon à ce que le nombre de réponses tardives soit affiché et qu'il intervienne dans l'appréciation.

8. Modifiez le programme précédent de façon à ce qu'il gère les erreurs liées à une faute de frappe (chaîne à la place d'un nombre, par exemple) avec les instructions `try` et `except` (voir le poly page 80).

Remarques

Pour générer un entier compris entre 1 et 10, on peut utiliser la fonction `randint` de la bibliothèque *random*. Quelques fonctions du module *random* sont évoquées page 303.
Pour déclencher un chronomètre, on peut utiliser la fonction `time` de la bibliothèque `time`. Pour l'arrêter, c'est la même fonction.
On peut également utiliser la fonction `clock` de la bibliothèque *time*. Le module *time* est détaillé dans le chapitre page 309.

L'affichage sera par exemple le suivant :

```
Appuyez sur Entrée pour démarrer
test 1 : 1 x 5 = 2
Dommage
test 2 : 2 x 7 = 14
bon mais trop tard : 5.61 secondes
test 3 : 4 x 8 = 32
Bravo !!!
test 4 : 3 x 6 = 18
Bravo !!!
test 5 : 5 x 8 = 40
Bravo !!!
test 6 : 5 x 6 = 30
Bravo !!!
test 7 : 2 x 9 = 18
Bravo !!!
test 8 : 10 x 9 = 90
Bravo !!!
test 9 : 5 x 6 = 30
Bravo !!!
test 10 : 4 x 10 = 40
Bravo !!!

Félicitations ! 8 bonne(s) réponse(s), 1 mauvaise(s) et 1 trop lente(s) sur 10

Une autre série ? (O/N) O

Appuyez sur Entrée pour démarrer
test 1 : 9 x 7 = 63
Bravo !!!
test 2 : 8 x 5 = 40
Bravo !!!
test 3 : 2 x 8 = 15
Dommage
test 4 : 4 x 7 = 28.5
Dommage
test 5 : 6 x 5 = hhj
could not convert string to float: 'hhj'
Dommage
test 6 : 2 x 7 = 13.75
Dommage
test 7 : 10 x 5 = 50
Bravo !!!
test 8 : 4 x 9 = 36
Bravo !!!
test 9 : 5 x 6 = 30
Bravo !!!
test 10 : 4 x 5 = 20
Bravo !!!

C'est moyen : 6 bonne(s) réponse(s), 4 mauvaise(s) et 0 trop lente(s) sur 10

Une autre série ? (O/N) N

14 de bonnes réponses sur 20, à bientôt
```

11

Solutions python

Partie 3

Travaux pratiques

Table des matières

12

Méthode de Newton

12.1 Compétences exigées

Objectifs

Les capacités évaluées dans cette partie de la formation sont :

- réaliser un programme complet structuré allant de la prise en compte de données expérimentales à la mise en forme des résultats permettant de résoudre un problème scientifique donné,
- étudier l'effet d'une variation des paramètres sur le temps de calcul, sur la précision des résultats, sur la forme des solutions pour des programmes d'ingénierie numérique choisis, tout en contextualisant l'observation du temps de calcul par rapport à la complexité algorithmique de ces programmes,
- utiliser les bibliothèques de calcul standard pour résoudre un problème scientifique mis en équation lors des enseignements de chimie, physique, mathématiques, sciences industrielles et de l'ingénieur,
- utiliser les bibliothèques standard pour afficher les résultats sous forme graphique,
- tenir compte des aspects pratiques comme l'impact des erreurs d'arrondi sur les résultats, le temps de calcul ou le stockage en mémoire.

12.2 Position du problème

En analyse numérique, la méthode de Newton, ou méthode de Newton-Raphson, est un algorithme efficace pour trouver des approximations d'un zéro (ou racine) d'une fonction d'une variable réelle à valeurs réelles. L'algorithme consiste à linéariser une fonction f en un point et à prendre le point d'annulation de cette linéarisation comme approximation du zéro recherché. On réitère cette procédure à partir de l'approximation obtenue.

Autrement dit, partant d'une valeur approximative d'une solution de l'équation $f(x) = 0$, on approxime la fonction par sa tangente en ce point. Cette tangente est une fonction affine dont on sait trouver l'unique zéro. Ce zéro de la tangente sera généralement plus proche du zéro de la fonction. En réitérant cette opération, on améliore l'approximation de la solution.

12.3 Méthode de résolution

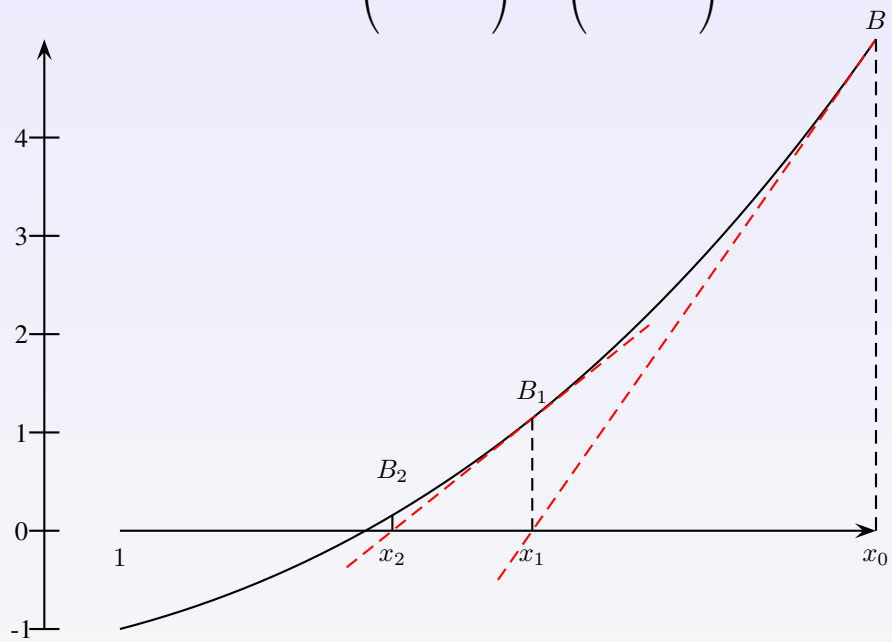
Principe

On souhaite par exemple approximer la solution de l'équation :

$$x^3 - x - 1 = 0$$

En partant de $x_0 = 2$, on obtient $y_{x_0} = 11x - 17$ pour l'équation de la tangente en 2, d'où $x_1 = \frac{17}{11}$, puis $y_{x_1} = \frac{746}{121}x - \frac{11157}{1331}$ d'où $x_2 = \frac{1157}{8206} \approx 0{,}14099$ et ainsi de suite.

La solution exacte de l'équation est : $\left(\frac{1+\sqrt{\frac{23}{27}}}{2}\right)^{\frac{1}{3}} + \left(\frac{1-\sqrt{\frac{23}{27}}}{2}\right)^{\frac{1}{3}}$



Plus formellement :

- Une abscisse x_n étant donnée, on cherche l'équation de la tangente à la courbe en ce point. Elle admet pour équation $y = f'(x_n)(x - x_n) + f(x_n)$.
- On cherche x_{n+1} qui est l'intersection de cette tangente avec l'axe des abscisses. On résout alors $f'(x_n)(x_{n+1} - x_n) + f(x_n) = 0$, ce qui donne :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- On réitère le procédé. On construit alors une suite (x_n) qui converge en général vers la solution.

En conclusion, on "résout" $f(x) = 0$ en formant la suite :

$$\begin{cases} x_0 &= \text{approximation de départ} \\ x_{n+1} &= x_n - \dfrac{f(x_n)}{f'(x_n)}. \end{cases}$$

⚠ La méthode de Newton ne fonctionne pas toujours. Il existe des conditions qui doivent être vérifiées : l'algorithme ne fonctionnera pas si la suite x_n tend vers l'infini (suite aboutissant sur l'abscisse d'un extremum) ou si la suite engendre un cycle.

12.4 Algorithme proposé

12.4.1 Premier algorithme

Pour l'algorithme, on définit la fonction :

$$g(x) = x - \frac{f(x)}{f'(x)}$$

L'algorithme associé à cette méthode est très court :

Newton

```
1   VARIABLES
2   x_0, x_n, précision : réels
3   Fonction f : x → f(x)
4   Fonction f' : x → f'(x)
5   DEBUT_ALGORITHME
6      INITIALISATION
7      x_n ← x_0
8      LIRE x_n, précision
9      TANT_QUE |f(x_n)| > précision FAIRE
10         DEBUT_TANT_QUE
11         x_n ← x_n - f(x_n)/f'(x_n)
12         solution ← x_n
13         FIN_TANT_QUE
14      AFFICHER solution
15
16  FIN_ALGORITHME
```

12.4.2 Amélioration de l'algorithme

Plusieurs soucis peuvent survenir :

- le nombre de boucles peut être très important dans des cas extrêmement défavorables et très rares : il serait donc bien de limiter le nombre d'itérations, par exemple avec un compteur (limité à 1000),
- si la dérivée de la fonction est nulle au départ ou bien si elle le devient en cours d'exécution, une erreur sera détectée (division par zéro !).

⇒ **Activité 12. 48**

Complétez l'algorithme précédent en conséquence.

Newton

```
1   VARIABLES
2   x_0, x_n, précision : réels
3   cpt : .....
4   Fonction f : x → f(x)
5   Fonction f' : x → f'(x)
6   DEBUT_ALGORITHME
7      INITIALISATION
8      x_n ← x_0
9      ..................
10      LIRE x_n, précision
11     TANT_QUE |f(x_n)| > précision ET ..... FAIRE
12        DEBUT_TANT_QUE
13        SI ..................... ALORS
14           DEBUT_SI
15           solution ← None
16           SORTIR de la boucle
17           FIN_SI
18        SINON
19           DEBUT_SINON
20           x_n ← x_n − f(x_n)/f'(x_n)
21           solution ← x_n
22           cpt ← cpt + 1
23           FIN_SINON
24        FIN_TANT_QUE
25     AFFICHER solution
26  FIN_ALGORITHME
```

12.5 Implémentation en python

12.5.1 Définition d'une fonction

Soit la fonction :

$$x^3 - x - 1 = 0$$

On peut définit cette fonction de 2 façons :

12.5.1.1 Avec *def*

Comme déjà vu, on peut le faire de la façon suivante, avec `def` :

```
def f(x):
  return x**3 - x - 1
```

12.5.1.2 Avec *lambda*

Il existe une autre méthode, avec `lambda`, lorsqu'on souhaite écrire une fonction simple :

```
f=lambda x: x**3-x-1
```

12.5.2 En fournissant l'expression de la dérivée

⇒ **Activité 12. 49**
Proposez, sous forme de fonction, le programme correspondant en *Python*. On prendra dans la fonction une valeur par défaut dans la précision de façon à ce que, si elle n'est pas précisée, elle vaille 10^{-6}.

12.5.3 En ne fournissant pas l'expression de la dérivée

On va supposer dans cette partie que nous ne savons pas trouver l'expression littérale de la dérivée de la fonction.

⇒ **Activité 12. 50**

- Proposez une fonction permettant de calculer en un point la dérivée d'une fonction. Si le pas de dérivation h n'est pas précisé par l'utilisateur, la fonction de va prendre une valeur par défaut, 10^{-5} par exemple.
- Proposez, sous forme de fonction, un nouveau programme, écrit en *Python*, permettant de calculer le zéro d'une fonction par la méthode de Newton. On prendra encore dans la fonction une valeur par défaut dans la précision de façon à ce que, si elle n'est pas précisée, elle vaille 10^{-6}.

Rappel

Définition de la dérivée d'une fonction :
Soit f une fonction définie sur un intervalle ouvert contenant x_0. On dit que f est dérivable en x_0 si la quantité $\frac{f(x_0+h)-f(x_0)}{h}$ admet une limite finie quand h tend vers 0. Cette limite est appelée nombre dérivé en x_0 et notée $f'(x_0)$.

$$\begin{aligned} f'(x) &= lim_{x\to x_0}\frac{f(x)-f(x_0)}{x-x_0} \\ &= lim_{h\to 0}\frac{f(x_0+h)-f(x_0)}{h} \end{aligned}$$

12.5.4 Avec *SciPy*

Des méthodes sont directement intégrées à la bibliothèque *scipy*. On peut le voir dans les programmes suivants :

12.5.5 En fournissant l'expression de la dérivée

- *newton2.py* :

newton2.py

```
1  # -*- coding: utf-8 -*-
2
3  import scipy as sp
4  import scipy.optimize as spop
5
6  def f(x):
7      return x**3-x-1
8  def f_prime(x):
9      return 3*x**2-1
10
11 print(format((((1+sp.sqrt(23/27))/2)**(1/3)+((1-sp.sqrt(23/27))/2)**(
                                                1/3)),'.25f'))
12 print(format(spop.newton(f,1,f_prime),'.25f'))
```

12.5.6 En ne fournissant pas l'expression de la dérivée

- *newton3.py* :

newton3.py

```
1  # -*- coding: utf-8 -*-
2
3  import scipy as sp
4  import scipy.optimize as spop
5
6  # pour changer : avec lambda
7  f=lambda x: x**3-x-1
8
9  print(format((((1+sp.sqrt(23/27))/2)**(1/3)+((1-sp.sqrt(23/27))/2)**(
                                                1/3)),'.25f'))
10 print(format(spop.newton(f,1),'.25f'))
```

12.6 Implémentation en *Scilab*

12.6.1 En fournissant l'expression de la dérivée

⇒ **Activité 12. 51**

Proposez, sous forme de fonction, le programme correspondant en *Scilab*.

Le programme :

12.6.2 En ne fournissant pas l'expression de la dérivée

On va supposer, ici aussi, que nous ne savons pas trouver l'expression littérale de la dérivée de la fonction.

⇒ **Activité 12. 52**

- Proposez une fonction permettant de calculer en un point la dérivée d'une fonction.

- Proposez, sous forme de fonction, un nouveau programme, écrit en *Scilab*, permettant de calculer le zéro d'une fonction par la méthode de Newton.

Le programme :

12.7 Comparaison des résultats

12.7.1 Avec *Python*

Programme *newton0.py* :

```
1.3247179572447458
1.3247181739990537
```

Programme *newton1.py* :

```
1.3247179572447458
1.3247181786548083
```

Programme *newton2.py* :

```
1.3247179572447458362205452
1.3247179572447460582651502
```

Programme *newton3.py* :

```
1.3247179572447458362205452
1.3247179572447529416479028
```

12.7.2 Avec *Scilab*

Programme *newton0.sce* :

```
1.3247179572447458

1.3247181739990537
```

Programme *newton1.sce* :

```
1.3247179572447458

1.3247182207076036
```

12.8 Comparaison des temps d'exécution

Sous *Python*, on peut par exemple utiliser le module *timeit* pour tester le temps d'exécution d'un programme.
L'exécution du programme suivant :

time_test_newton.py

```
1  # -*- coding: utf-8 -*-
2
3  import timeit
4
5  t=timeit.Timer('import newton0','')
6  print("newton0 : ",t.timeit())
7
8  print()
9  t=timeit.Timer('import newton1','')
10 print("newton1 : ",t.timeit())
11
12 print()
13 t=timeit.Timer('import newton2','')
14 print("newton2 : ",t.timeit())
15
16 print()
17 t=timeit.Timer('import newton3','')
18 print("newton3 : ",t.timeit())
```

donne :

```
1.3247179572447458
1.3247181739990537
newton0 :  0.285410115000559

1.3247179572447458
1.3247181786548083
newton1 :  0.22631195500071044

1.3247179572447458362205452
1.3247179572447460582651502
newton2 :  0.27091168600236415

1.3247179572447458362205452
1.3247179572447529416479028
newton3 :  0.22385916900020675
```

12.9 Tracé de la fonction

⇒ **Activité 12. 53**

Pour ceux qui ont le temps, il est intéressant d'effectuer le tracé de la fonction $x^3 - x - 1$.

Cette courbe se fera par exemple à l'aide de la sous-bibliothèque *pyplot* de *matplotlib* ...

Pour utiliser `linspace`, il faudra importer la bibliothèque *numpy*.

Le programme pourra être enregistré sous le nom de *newton-graphe.py*.

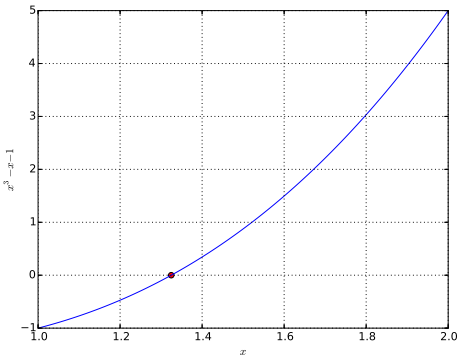


FIGURE 12.1 – newton-graphe

Partie 4

Python année 2

Table des matières

13

Traitement d'image

Objectifs

Les capacités évaluées dans cette partie de la formation sont :

- programmer un algorithme dans un langage de programmation moderne et général,
- modifier un algorithme existant pour obtenir un résultat différent,
- concevoir un algorithme répondant à un problème précisément posé,
- expliquer le fonctionnement d'un algorithme,
- comprendre le fonctionnement d'un algorithme récursif et l'utilisation de la mémoire lors de son exécution,
- comprendre les avantages et défauts respectifs des approches récursive et itérative,
- s'interroger sur l'efficacité algorithmique temporelle d'un algorithme,
- distinguer par leurs complexités deux algorithmes résolvant un même problème.

13.1 Généralités

- Un *bit* (abréviation de *binary digit*, signifiant chiffre binaire en français) correspond à la plus petite unité élémentaire de stockage d'informations en informatique, soit un chiffre binaire 0 ou 1. Le symbole de bit est b.
- Un *byte* signifie *octet* en français. Un *octet* (*byte*) est constitué de 8 bits. Le symbole de byte est B.
- Le symbole de *octet* est o.

Remarque

Il ne faut donc pas confondre les unités de stockage :

1 B	= 1 o	= 8 b
1 kB	= 1 ko	= 8 kb
1 MB	= 1 Mo	= 8 Mb

Une image matricielle est aussi appelée image ponctuelle.

Un "bitmap" est un rectangle de "pixels" (*pixel* pour l'abréviation de *picture element*), donc une matrice.

Une image en "Noir et Blanc" contient des pixels qui sont codés sur 1 bit : 0 ou 1 (ou encore 0 ou 255).

Une image en niveaux de gris contient des pixels qui sont eux-mêmes codés sur 1 octet = 8 bits, c'est-à-dire de 0 à 255 : 256 = 2^8. Cette image en niveaux de gris est donc une liste de nombres compris entre 0 et 255. Le nombre d'éléments de cette liste est égal au nombre de pixels, 64 par exemple pour une image de 8 x 8.

Une image en couleurs contient des pixels contenant, chacun, une liste de 3 nombres compris entre 0 et 255, chaque liste correspondant à 3 couleurs : rouge, vert et bleu pour le mode "RGB" (red, green, blue). Une image en couleurs est donc une liste de listes de 3 termes.

Chaque pixel est alors codé sur 3 x 8 = 24 bits = 3 octets. Cela correspond à un choix de :
$\left(2^8\right)^3 = 2^{24} = 16777216$ couleurs, soit environ 16,7 millions de couleurs.

Définition

La définition d'une image est le nombre de pixels total, c'est-à-dire sa "dimension informatique" (le nombre de colonnes de l'image multiplié par son nombre de lignes). Ainsi, une image possédant 3000 pixels en largeur et 2000 en hauteur aura une définition de 3000 pixels par 2000, notée 3000 x 2000 = 6 Mpx. Puisqu'un pixel est codé 3 octets, cette image pèse 18 Mo (1 $Mo = 10^6$ octets).

Résolution

La résolution d'une image composée de points est définie par la densité des points par unité de surface. La résolution permet de définir la finesse de l'image. Plus la résolution est grande, plus la finesse de l'image est grande.
Sur les écrans on parle de "pixel" alors que sur le papier, on parle de "points" où "dots".
Ainsi, la résolution dans le domaine de l'écran est *ppi* (pixels per inch : ppp en français, soit pixels par pouce) alors que la résolution sur le papier est *dpi* (dots per inch : points par pouce) : 1 pouce (*inch* en anglais) = 2,54 cm.

On a la relation : $\boxed{\text{résolution} = \dfrac{\text{définition}}{\text{dimension}}}$

Il existe beaucoup de possibilités pour réaliser du traitement d'image sous *Python*. Par exemple, sous *MatPlotLib*, certaines fonctionnalités permettent de traiter les images. Il existe aussi des bibliothèques dédiées au traitement d'images de niveau professionnel avec lesquelles on peut travailler sous *Python* (par exemple OpenCV [1]).

Dans un premier temps, nous passerons en revue quelques opérations réalisables sous *PIL* ainsi que *Scipy* et *Matplotlib*. Ensuite, nous verrons comment traiter les images en parcourant les pixels, écrits sous forme de matrices.

L'avantage principal de la bibliothèque *PIL* est sa simplicité de mise œuvre.

On pourra importer le module *Image* de la bibliothèque *PIL* par la commande suivante :

```
from PIL import Image
```

⚠ Il faudra éviter les conflits avec d'autres bibliothèques (*tkinter* par exemple). En cas de message d'insultes de la part de *Python*, il faudra penser à ce type d'erreur possible.

13.2 Le module *PIL*

13.2.1 Installation du module

1. http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html

13.2.1.1 Sous *Ubuntu*

En principe, les commandes en console *linux* doivent fonctionner :

```
sudo apt-get install python3-setuptools

sudo easy_install3 pip

sudo apt-get install python3-dev

 wget https://github.com/python-imaging/Pillow/archive/master.zip

sudo unzip master.zip

ls


ls*

cd Pillow-master/

sudo python3 setup.py build

sudo python3 setup.py install

exit

sudo apt-get install libtiff4-dev libjpeg8-dev zlib1g-dev \
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.5-dev tk8.5-dev python-tk

cd Pillow-master/

sudo python3 setup.py build

cd ..

sudo apt-get install python3-pip

sudo pip install -I pillow
```

13.2.1.2 Sous *Windows*

En principe, le téléchargement et l'installation du fichier téléchargeable ici :

https://pypi.python.org/pypi/Pillow/2.5.3

doit suffire.

Il faut aller au bas de la page et choisir la version qui vous convient parmi les fichiers suivants :

- Pillow-2.5.3-py3.2-win-amd64.egg (md5)
- Pillow-2.5.3-py3.2-win32.egg (md5)
- Pillow-2.5.3-py3.3-win-amd64.egg (md5)
- Pillow-2.5.3-py3.3-win32.egg (md5)
- Pillow-2.5.3-py3.4-win-amd64.egg (md5)
- Pillow-2.5.3-py3.4-win32.egg (md5)

Vous trouverez la version pour *Python 3.3* dans le dossier d'installation.

13.2.2 Ouverture d'un fichier image

La méthode `open(chemin)` permet d'ouvrir une image. Les formats supportés par la bibliothèque *PIL* sont :

- "BMP" : extension *.bmp* ou *.dib*,
- "DCX" : extension *.dcx*,
- "EPS" : extension *.eps* ou *.ps*,
- "GIF" : extension *.gif*,
- "IM" : extension *.im*,
- "JPEG" : extension *.jpg*, *.jpe* ou *.jpeg*,
- "PCD" : extension *.pcd*,
- "PCX" : extension *.pcx*,
- "PDF" : extension *.pdf*,
- "PNG" : extension *.png*,
- "PPM" : extension *.pbm*, *.pgm* ou *.ppm*,
- "PSD" : extension *.psd*,
- "TIFF" : extension *.tif* ou *.tiff*,
- "XBM" : extension *.xbm*,
- "XPM" : extension *.xpm*.

La méthode `getdata()` retourne un objet-séquence contenant les valeurs des pixels de l'image. Cependant il n'est lisible que par *PIL*. La fonction `list()` permet ensuite de récupérer cette séquence sous un format lisible par l'utilisateur. On récupère alors une liste de tuples à 3 composantes si l'image est couleur, une liste simple sinon.

```
mon_image = Image.open("tux20.jpg")
donnees = list(mon_image.getdata())
```

Exemple avec le programme suivant :

data1.py

```
1 from PIL import Image
2
3 mon_image = Image.open("test.png")
4 donnees = list(mon_image.getdata())
5 print(donnees)
```

qui donne :

```
[(109, 148, 114), (109, 148, 114), (109, 148, 114), (167, 57, 51),
(109, 148, 114), (109, 148, 114), (109, 148, 114), (109, 148, 114),
(239, 226, 21), (109, 148, 114), (109, 148, 114), (109, 148, 114)]
```

13.2.3 Visualisation d'une image

Il faut utiliser la méthode `show()` :

```
mon_image.show()
```

13.2.4 Taille d'une image

taille

La taille d'une image correspond à son nombre de colonnes (largeur) et son nombre de lignes (hauteur).

La fonction `size` permet d'accéder à ce tuple (largeur, hauteur) :

```
largeur,hauteur = mon_image.size
```

On obtient ici :

```
4 3
```

Remarque

`largeur` correspond à `mon_image.size[0]` et `hauteur` à `mon_image.size[1]`

13.2.5 Bounding box

Bounding box

Une *bounding box* est un rectangle contenu dans l'image. Elle est définie par un tuple de 4 termes (x_0, y_0, x_1, y_1). (x_0, y_0) sont les coordonnées du point qui est en haut à gauche alors que(x_1, y_1) correspondent à celles du coin en bas à droite.

Cela peut servir par exemple à extraire ou copier une partie de l'image.

13.2.6 Autres propriétés de l'image

On peut accéder à d'autres propriétés de l'image à l'aide des commandes `format`, `mode`, `palette` et `info`. Ainsi, les instructions suivantes :

```
print(mon_image.format)
print(mon_image.mode)
print(mon_image.palette)
print(mon_image.info)
```

donnent :

```
JPEG
RGB
None
{'jfif_density': (35, 35), 'jfif_version': (1, 1), 'jfif': 257, 'jfif_unit': 2}
```

Le mode définit la façon dont sont définies les couleurs. Chaque mode est représenté par une chaîne.
Le nombre de bandes est un ensemble de valeurs qui définit un pixel (1 bande pour le noir et blanc, 3 bandes pour le mode "RGB", ...). Les principaux modes sont :

Mode	bandes	Description
"1"	1	Noir et blanc (monochrome) : 1 bit par pixel
"L"	1	Niveaux de gris : 1 byte de 8 bits par pixel
"RGB"	3	Mode couleurs RGB : 3 bytes de 8 bits par pixel
"RGBA"	4	Mode couleurs RGB avec une bande de transparence A : 4 bytes par pixel. Le canal A varie de 0 pour transparent à 255 pour opaque
"CMYK"	4	Mode Cyan - Magenta - Yellow - Black : 4 bytes par pixel
"YCbCr"	3	Format de couleur spécial : [2]3 bytes de 8 bit par pixel
"I"	1	pixels 32 bit entier
"F"	1	pixels 32 bit flottant

TABLE 13.1 – Modes d'une image

13.2.7 Création d'une image à partir d'une liste de valeurs de pixels

La méthode `new(mode, taille)` permet de créer une nouvelle image où *mode* ("L" ou "RGB", par exemple) définit si l'image sera en niveaux de gris ou en couleurs et *taille* est un tuple (*nombre de colonnes*, *nombre de lignes*), donc (*largeur*, *hauteur*).
La méthode `putdata(données)` remplit l'image avec la séquence de valeurs *données*.
Dans l'exemple suivant, on reconstitue une image identique à l'originale :

```
ma_nouvelle_image=Image.new('RGB',(largeur,hauteur))
ma_nouvelle_image.putdata(donnees)
ma_nouvelle_image.show()
```

13.2.8 Décomposition d'une image couleur en ses 3 composantes *RGB*

La méthode `split()` renvoie une séquence des 3 composantes RGB de l'image :

```
## Récupération des différentes composantes de l'image
r,g,b = ma_nouvelle_image.split()
## Sauvegarde des différentes composantes de l'image
r.save('red.jpg')
g.save('green.jpg')
b.save('blue.jpg')
## Visualisation de la composante rouge
Image.open('red.jpg').show()
```

(a) rouge (b) vert (c) bleu

FIGURE 13.1 – Niveaux de rouge, vert et bleu de l'image originale

2. http://fr.wikipedia.org/wiki/YCbCr

13.2.9 Composition d'une image à partir des ses 3 composantes *RGB*

La méthode `merge(mode,bands)` crée une image à partir des images *bands* selon le mode précisé :

```
## décomposition de l'image
ma_composition=r,g,b
 ## Recomposition de l'image
mon_image_recomposee = Image.merge('RGB',ma_composition)
## Sauvegarde de l'image décomposée puis recomposée
mon_image_recomposee.save("tux20c.jpg")
## Visualisation de l'image recomposée
Image.open('tux20c.jpg').show()
```

Les différentes instructions qui précèdent sont intégrées au programme *ouverture.py* : celui-ci est imprimé page 165.

13.2.10 Conversion de format

Avec *PIL*, il est possible de convertir une image sous un autre format, par exemple : "png", "jpeg", "bmp", "eps", ... avec la méthode `save()`.

```
## Conversion de format
r.save('red.eps')
g.save('green.eps')
b.save('blue.eps')
```

13.2.11 Permutation des bandes

Une fois que l'on a accès aux bandes "RGB", il est facile de les permuter. Exemple avec le programme suivant :

permute_bandes.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  mon_image = Image.open("billes.jpg")
6  mon_image.save("billes.eps")
7
8  def permute_bandes1(im):
9      """permutation circulaire des composantes RGB"""
10     rouge,vert,bleu = im.split()
11     return Image.merge("RGB",(bleu,rouge,vert))
12
13 def permute_bandes2(im):
14     """permutation circulaire des composantes RGB"""
15     rouge,vert,bleu = im.split()
16     return Image.merge("RGB",(vert,bleu,rouge))
17
18 nouvelle_image1 = permute_bandes1(mon_image)
19 nouvelle_image1.save('billes_perm1.eps')
20 nouvelle_image1.show()
21
22 nouvelle_image2 = permute_bandes2(mon_image)
23 nouvelle_image2.save('billes_perm2.eps')
24 nouvelle_image2.show()
```

Le résultat :

(a) billes (b) billes_perm1 (c) billes_perm2

FIGURE 13.2 – Permutation des bandes

13.2.12 Conversion de mode

Avec *PIL*, il est également possible de convertir une image dans un autre mode avec la méthode `convert()`, par exemple : "L" pour niveaux de gris, "1" pour monochrome, ...

```
## Conversion de mode
image_gris = mon_image.convert('L')
image_gris.save('tux20_gris.eps')

image_NetB = mon_image.convert('1')
image_NetB.save('tux20_NetB.bmp')
```

On obtient :

(a) couleurs RGB (b) niveaux de gris (c) monochrome

FIGURE 13.3 – Niveaux de gris et monochrome

13.2.13 Image miniature

Pour créer une miniature à partir de l'image originale, on peut utiliser la méthode `resize(taille,resample)`. La *taille* est un tuple (largeur, hauteur) et *resample* peut être précisé pour choisir le mode d'interpolation. Par défaut, c'est `NEAREST` qui est choisi. Les 4 filtres applicables sont :

- NEAREST,
- BILINEAR,
- BICUBIC,
- ANTIALIAS.

```
## Miniature
retaillee=mon_image.resize((128,128),resample=Image.BILINEAR)
retaillee.show()
```

13.2.14 Opérations sur un répertoire

Pour réaliser par exemple l'opération précédente sur tout un répertoire, on peut utiliser le programme :

operation_rep.py

```
1  # -*- coding: utf-8 -*-
2  from PIL import Image
3  import glob, os
4
5  size = (128, 128)
6
7  for fichier in glob.glob("*.png"):
8      nom_fichier, ext = os.path.splitext(fichier)
9      mon_image = Image.open(fichier)
10     mon_image.thumbnail(size, Image.ANTIALIAS)
11     mon_image.save(nom_fichier + "_small.png", "PNG")
```

Bien sûr, l'importation de *glob* et de *os* peut servir à bien d'autres utilisations : ces modules permettent d'explorer et manipuler les répertoires, sous-répertoires ainsi que les fichiers.

13.2.15 Transformations isométriques

On peut réaliser une rotation à l'aide de la fonction `rotate()` et réaliser d'autres transformations à l'aide de la fonction *transpose(méthode)*. Ces opérations sont visibles dans le programme suivant :

rotation.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  mon_image = Image.open('tux20.jpg')
6  sortie = mon_image.rotate(45)
7  sortie.save('tourne.jpg')
8  sortie.save('tux20_rot45.eps')
9  sortie.show()
10
11 sortie = mon_image.transpose(Image.FLIP_LEFT_RIGHT)
12 sortie.save('tux20_flip_left_right.eps')
13 sortie.show()
14
15 sortie = mon_image.transpose(Image.FLIP_TOP_BOTTOM)
16 sortie.save('tux20_flip_top_bottom.eps')
17 sortie.show()
18
19 sortie = mon_image.transpose(Image.ROTATE_270)
20 sortie.save('tux20_rotate_270.eps')
21 sortie.show()
```

On obtient les images suivantes :

(a) rotate(45) (b) FLIP_LEFT_RIGHT (c) FLIP_TOP_BOTTOM (d) ROTATE_270

FIGURE 13.4 – rotate et transpose

Les opérations possibles avec `transpose()` sont :

- sortie = *mon_image.transpose(Image.FLIP_LEFT_RIGHT)*
- sortie = *mon_image.transpose(Image.FLIP_TOP_BOTTOM)*
- sortie = *mon_image.transpose(Image.ROTATE_90)*
- sortie = *mon_image.transpose(Image.ROTATE_180)*
- sortie = *mon_image.transpose(Image.ROTATE_270)*

13.2.16 Filtrage

Le filtrage d'une image se réalise à l'aide de la fonction `filter()` de la sous-bibliothèque *ImageFilter* qui propose de nombreux filtres (filtres min, max, médian, flou, ...). La liste des filtres est indiquée ci-dessous :

- ImageFilter.BLUR,
- ImageFilter.CONTOUR,
- ImageFilter.DETAIL,
- ImageFilter.EDGE_ENHANCE,
- ImageFilter.EDGE_ENHANCE_MORE,
- ImageFilter.EMBOSS,
- ImageFilter.FIND_EDGES,
- ImageFilter.SMOOTH,
- ImageFilter.SMOOTH_MORE,
- ImageFilter.SHARPEN.

Des exemples sont présentés dans le programme suivant dont un utilisant `ImageChops` :

filtres.py

```
1  #-*- coding: utf-8 -*-
2
3  from PIL import Image
4  from PIL import ImageChops
5  from PIL import ImageFilter
6
7  Im = Image.open('billes.jpg')
8  # print(Im.format, Im.size, Im.mode)
9
10 # binarisation
11 ImBW = Im.convert('1')
12 ImBW.save('billes_BW.bmp')
13 ImBW.show()
14
15 # niveaux de gris
16 ImL = Im.convert('L')
17 ImL.save('billes_gris.eps')
18 ImL.show()
19
20 # inversion d'image (négatif)
21 ImInv = ImageChops.invert(Im)
22 ImInv.save('billes_Inv.eps')
23 ImInv.show()
24
25 # BLUR filter
26 ImBlur = Im.filter(ImageFilter.BLUR)
```

```
27  ImBlur.save('billes_BLUR.eps')
28  ImBlur.show()
29
30  # CONTOUR filter
31  ImContour = Im.filter(ImageFilter.CONTOUR)
32  ImContour.save('billes_CONTOUR.eps')
33  ImContour.show()
34
35  # DETAIL filter
36  ImDetail = Im.filter(ImageFilter.DETAIL)
37  ImDetail.save('billes_DETAIL.eps')
38  ImDetail.show()
39
40  # EDGE_ENHANCE filter
41  ImEH = Im.filter(ImageFilter.EDGE_ENHANCE)
42  ImEH.save('billes_EDGE_ENHANCE.eps')
43  ImEH.show()
44
45  # EDGE_ENHANCE_MORE filter
46  ImEHM = Im.filter(ImageFilter.EDGE_ENHANCE_MORE)
47  ImEHM.save('billes_EHM.eps')
48  ImEHM.show()
49
50  # EMBOSS filter
51  ImEmb = Im.filter(ImageFilter.EMBOSS)
52  ImEmb.save('billes_EMBOSS.eps')
53  ImEmb.show()
54
55  # FIND_EDGES filter
56  ImFed = Im.filter(ImageFilter.FIND_EDGES)
57  ImFed.save('billes_EDGES.eps')
58  ImFed.show()
59
60  # SMOOTH filter
61  ImSm = Im.filter(ImageFilter.SMOOTH)
62  ImSm.save('billes_SMOOTH.eps')
63  ImSm.show()
64
65  # SMOOTH_MORE filter
66  ImSmm = Im.filter(ImageFilter.SMOOTH_MORE)
67  ImSmm.save('billes_SMOOTH_MORE.eps')
68  ImSmm.show()
69
70  # SHARPEN filter
71  ImSh = Im.filter(ImageFilter.SHARPEN)
72  ImSh.save('billes_SHARPEN.eps')
73  ImSh.show()
74
75  # MinFilter filter
76  ImMin = Im.filter(ImageFilter.MinFilter)
77  ImMin.save('billes_MinFilter.eps')
78  ImMin.show()
79
80  # MedianFilter filter
81  ImMed = Im.filter(ImageFilter.MedianFilter)
82  ImMed.save('billes_MedianFilter.eps')
83  ImMed.show()
84
85  # MaxFilter filter
86  ImMax = Im.filter(ImageFilter.MaxFilter)
```

```
87  ImMax.save('billes_MaxFilter.eps')
88  ImMax.show()
```


et le résultat :

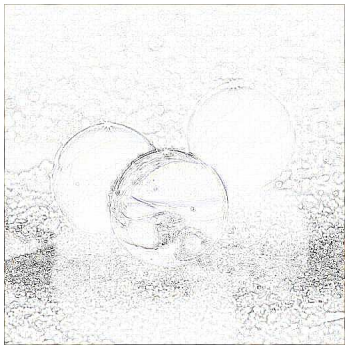
(a) original

(b) convert('1')

(c) convert('L')

(d) Invert

(e) BLUR

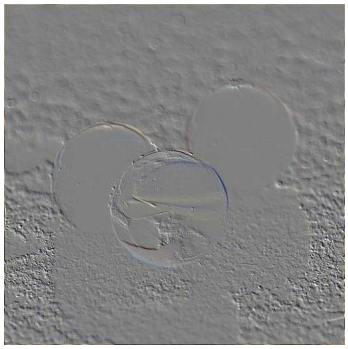


(f) CONTOUR

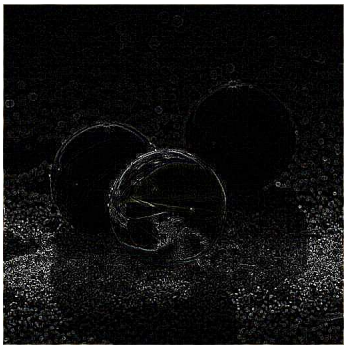
(g) DETAIL

(h) EDGE_ENHANCE

(i) EHM



(j) EMBOSS



(k) FIND_EDGES

(l) SMOOTH

(m) SMOOTH_MORE (n) SHARPEN (o) MinFilter

(p) MedianFilter (q) MaxFilter

FIGURE 13.5 – ImageFilter

Remarque

`ImageChops` permet de comparer deux images, de les soustraire (`difference()`), de les additionner (`add()`), de les copier (`duplicate`), de les surimprimer, de les comparer, ...

Vous pouvez tester si vous voulez, le programme suivant :

addition-image.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4  from PIL import ImageChops
5
6  tour = Image.open('tourne.jpg')
7  tux20 = Image.open('tux20.jpg')
8
9  res = ImageChops.add(tour,tux20,1,0)
10 res.save('addition.eps')
11 res.show()
```

qui donne :

FIGURE 13.6 – Addition

13.2.17 Autres Exemples de manipulations

13.2.17.1 Concaténation d'images

Pour concaténer une image avec son symétrique, on peut regarder le programme ci-dessous :

miroir.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4  mon_image = Image.open("tux20.jpg")
5
6  largeur,hauteur=mon_image.size
7  box = (0, 0, largeur, hauteur)
8  src = mon_image.crop(box)
9  sortie = mon_image.resize((largeur*2,hauteur))
10 sortie.paste(src,(0,0,largeur,hauteur))
11 src=mon_image.transpose(Image.FLIP_LEFT_RIGHT)
12 sortie.paste(src,(largeur,0,2*largeur,hauteur))
13 sortie.save('tux20_miroir.eps')
14 sortie.show()
```

FIGURE 13.7 – Image et symétrique

13.2.17.2 Dessiner sur une image

Le module de dessin de *PIL*, *ImageDraw*, contient différentes méthodes. On peut en citer quelques unes :

- `arc (bbox, début, fin, couleur de trait)` (par défaut, la couleur est `white`) qui trace un arc de cercle,
- `ellipse (bbox, couleur intérieure, couleur de trait)` pour représenter une ellipse,
- `line (points extrêmes, couleur de trait)`,
- `point ((x, y), couleur)` pour représenter un point,
- `polygon (points extrêmes, couleur intérieure, couleur de trait)` pour tracer un polygone,,
- `text ((x, y), message, couleur de trait, police)` pour écrire un texte de coordonnées (x, y) au coin haut à gauche.

Certaines nécessitent une *bounding box* (cf. 13.2.5 page 153).

Quelques fonctions sont illustrées dans le programme suivant :

dessin.py

```
1  # -*- coding: utf-8 -*-
2
```

```
3  from PIL import Image
4  from PIL import ImageDraw
5
6  mon_image = Image.new("RGB", (400,200), "lightgrey")
7  draw = ImageDraw.Draw(mon_image)
8  largeur,hauteur=mon_image.size
9  draw.ellipse((largeur/4,hauteur/4) + (3*largeur/4,3*hauteur/4), fill=
                                          "blue")
10 draw.line((0, 0) + mon_image.size, width=8,fill="red")
11 offsetx=100
12 draw.rectangle(((80+offsetx,100),(130+offsetx,200)),fill="gray")
13 draw.ellipse(((largeur/3),(hauteur/3),(largeur*2/3),(hauteur*2/3)),
                                          fill="white")
14 del draw
15 mon_image.save("dessin.eps", "EPS")
16 mon_image.show()
```

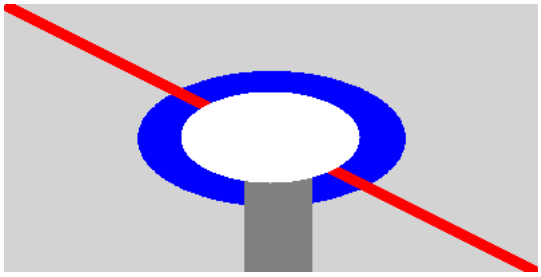


FIGURE 13.8 – dessin

13.2.17.3 Image multiple

pil_multi.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4  from PIL import ImageDraw
5
6  mon_image = Image.open("tux20.jpg")
7  nouvelle_image = Image.new("RGB", (530,350), "white")
8  draw = ImageDraw.Draw(mon_image)
9  largeur,hauteur=mon_image.size
10 nouvelle_image.paste(mon_image, (0,0,largeur,hauteur))
11 nouvelle_image.paste(mon_image, (300,0,300+largeur,hauteur))
12 nouvelle_image.paste(mon_image, (200,200,200+largeur,200+hauteur))
13 del draw
14 nouvelle_image.save("pil_multi.eps")
15 nouvelle_image.show()
```

Le résultat :

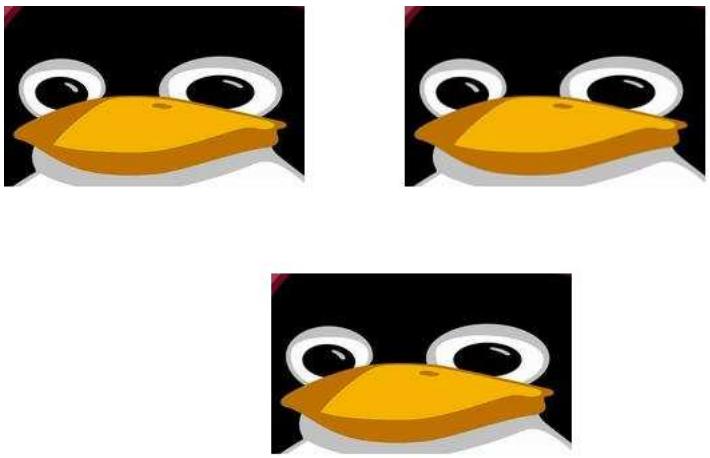


FIGURE 13.9 – pil_multi.eps

13.2.17.4 Méthodes d'un objet image

Certaines méthodes ont déjà été abordées, d'autres non. Voici quelques méthodes applicables sur un objet Image :

méthode	description
`save(fichier, format=None)`	Sauvegarde l'image dans un fichier. Si fichier est un nom de fichier, le format est choisi automatiquement
`convert(mode)`	Renvoie une nouvelle image dans un mode différent
`copy()`	Renvoie une copie de l'image
`crop(bbox)`	Renvoie une nouvelle image constituée par la *bounding box* : cf. 13.2.5 page 153 (spécifiée en argument) correspondante de l'image d'origine
`filter(nom)`	Renvoie une copie de l'image, filtrée avec le filtre d'amélioration du nom de *nom* : cf 13.2.16 page 158
`getbands()`	Renvoie une chaîne de caractères, une par bande, représentant le mode de l'image
`getbox()`	Renvoie la plus petite bounding box qui enferme la partie non nulle de l'image
`getdata()`	Renvoie le contenu d'une image dans une suite contenant les valeurs des pixels. La suite est "étalée" : les valeurs de la seconde ligne suivent immédiatement celles de la première, ...
`getextrema()`	Dans le cas des images simple-bande, renvoie un couple (`min`, *max*), où `min` est la plus petite valeur, et `max` la plus grande valeur, des pixels de l'image. Dans le cas des images multi-bandes, renvoie le tuple des couples (`min`, `max`) pour chaque bande
`getpixel((x,y))`	Retourne la (ou les) valeur(s) du pixel de coordonnées (x, y).
`histogram(masque=None)`	Pour les images simples-bandes, renvoie une liste de valeurs [c_0, c_1, ...], où c_i est le nombre de pixels ayant la valeur i. Pour les multi-bandes, les différentes listes de valeurs (pour chaque bande) sont concaténées (masque facultatif)
`offset(dx,dy)`	*dy* est facultatif
`paste(mon_image,bbox, mask = None)`	Remplace les pixels par ceux de *mon_image* (masque facultatif)
`point(fonction)`	Permet de modifier une image pixel par pixel, grâce à une `fonction` de transformation.

méthode	description
`point(tableau)`	Permet de transformer les pixels suivant un `tableau`. Le `tableau` doit être une suite de $256\,n$ valeurs, où n est le nombre de bandes de l'image. Chaque pixel de la bande b de l'image est remplacé par la valeur de `tableau[p+256*b]`, où p est l'ancienne valeur du pixel dans la bande considérée.
`putalpha(bande)`	Ajoute une bande alpha (transparence) à une image de mode "RGBA"
`putpixel((x,y), couleur)`	Remplace le pixel $(x,\ y)$ de l'image. Pour les images multi-bandes, la valeur est un tuple
`resize(taille,resample)`	Renvoie une nouvelle image ayant la taille spécifiée, en procédant linéairement (*resample* facultatif)
`rotate(`θ`)`	Renvoie une image qui a tournée d'un angle θ, en degrés, par rapport au centre de l'image considérée. Le sens de la rotation est trigonométrique
`show()`	Affiche l'image
`split()`	Renvoie un tuple contenant chaque bande de l'image originale, vue comme une image de mode "L". Par exemple, appliquer cette méthode à une image "RGB" produit un triplet d'images, la première pour la bande rouge, la seconde pour la verte et la dernière pour la bleue
`thumbnail(taille,filter=None)`	Remplace l'image originale par une nouvelle image ayant la taille *taille*. Le filtre optionnel fonctionne comme pour la méthode `resize()`
`transform(`x_s`, `y_s`, Image.EXTENT, (`x_0`, `y_0`, `x_1`, `y_1`))`	Renvoie une transformation de l'image : le point originellement en (x_0, y_0) se retrouvera en $(0, 0)$, et le point (x_1, y_1) en (x_s, y_s).
`transpose(méthode)`	Renvoie une copie, culbutée ou retournée, de l'image originale

TABLE 13.3 – Méthodes d'un objet image

Voici le programme *ouverture.py* pour vous repérer éventuellement :

ouverture.py

```
1  # -*- coding: utf-8 -*-
2
3  ## Import du module PIL
4  from PIL import Image
5
6  ## Ouverture de l'image
7  mon_image = Image.open("tux20.jpg")
8  ## sauvegarde de l'image en eps
9  mon_image.save("tux20.eps")
10 ## sauvegarde de l'image en png
11 mon_image.save("tux20.png")
12 ## Visualisation de l'image
13 mon_image.show()
14
15 ## Matrice image
16 donnees = list(mon_image.getdata())
17 print(donnees)
18
19 ## Taille de l'image
20 largeur,hauteur = mon_image.size
21 print(largeur,hauteur)
22
23 ## Propriétés
```

```
24 print(mon_image.format)
25 print(mon_image.mode)
26 print(mon_image.palette)
27 print(mon_image.info)
28
29 ma_nouvelle_image=Image.new('RGB',(largeur,hauteur))
30 ma_nouvelle_image.putdata(donnees)
31 ma_nouvelle_image.show()
32
33 ## Récupération des différentes composantes de l'image
34 rouge,vert,bleu = ma_nouvelle_image.split()
35 ## Sauvegarde des différentes composantes de l'image
36 rouge.save('red.jpg')
37 vert.save('green.jpg')
38 bleu.save('blue.jpg')
39 ## Visualisation de la composante rouge
40 Image.open('red.jpg').show()
41
42 ## décomposition de l'image
43 ma_composition=rouge,vert,bleu
44  ## Recomposition de l'image
45 mon_image_recomposee = Image.merge('RGB',ma_composition)
46 ## Sauvegarde de l'image décomposée puis recomposée
47 mon_image_recomposee.save("tux20c.jpg")
48 ## Visualisation de l'image recomposée
49 Image.open('tux20c.jpg').show()
50
51 ## Transformations
52
53 ## Conversion de format
54 rouge.save('red.eps')
55 vert.save('green.eps')
56 bleu.save('blue.eps')
57
58 ## Conversion de mode
59 image_gris = mon_image.convert('L')
60 image_gris.save('tux20_gris.png')
61 image_gris.save('tux20_gris.eps')
62
63 image_NetB = mon_image.convert('1')
64 image_NetB.save('tux20_NetB.bmp')
65
66 ## Miniature
67 retaillee=mon_image.resize((128,128),resample=Image.BILINEAR)
68 retaillee.show()
```

13.3 *Scipy* et les images

Le module *Scipy* aidé de *matplotlib* permet également d'effectuer des traitements sur les images.

Les manipulations sont un peu compliquées et assez inintéressantes du point de vue de la programmation. On n'y consacrera donc pas de temps.

Vous pourrez trouver des ressources ici :
http://perso.telecom-paristech.fr/~gramfort/liesse_python/3-Scipy.pdf

13.4 Manipulation des fichiers images

Dans cette partie, il s'agira, pour modifier une image, de parcourir sa matrice.

13.4.1 Introduction

Voici le programme *image-array.py* :

image-array.py

```
1  #-*- coding: utf-8 -*-
2
3  from PIL import Image
4  import numpy as np
5
6  image1 = Image.open('tux20_NetB.bmp')
7  largeur, hauteur = image1.size
8  print(largeur,hauteur,'\n')
9
10 donnees = list(image1.getdata())
11 print(donnees[0:30],'\n')
12
13 tableau = np.array(donnees)
14 print(tableau,'\n')
15 print(np.shape(tableau),'\n')
16
17 matrice = np.reshape(tableau,(hauteur,largeur))
18 # attention matrice = lignes * colonnes
19 # alors que donnees = largeur * hauteur
20 print(matrice,'\n')
21 print(np.shape(matrice),'\n')
22
23 matrice_etalee = list(matrice.flat)
24 print(matrice_etalee[0:30],'\n')
25
26 image2 = Image.new('L',(matrice.shape[1],matrice.shape[0]))
27 image2.putdata(matrice_etalee)
28 image2.save('tux20_reconstruit.eps')
29 image2.show()
```

ainsi que le résultat en console :

```
225 135

[0, 255, 0, 0, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[  0 255   0 ..., 255 255 255]

(30375,)

[[  0 255   0 ...,   0   0   0]
 [  0   0 255 ...,   0   0   0]
 [255   0   0 ...,   0   0   0]
 ...,
 [  0   0   0 ..., 255 255 255]
 [  0   0   0 ..., 255 255 255]
 [  0   0   0 ..., 255 255 255]]

(135, 225)

[0, 255, 0, 0, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

⇒ **Activité 13. 54**

Commentez chaque ligne du programme en fonction du résultat donné. Il est très important de bien comprendre cela pour la suite car même si on pourra éviter de passer par les matrices (type `array`), l'idée sera la même.

Par la suite, il sera parfois plus facile de parcourir les matrices images et de travailler pixel par pixel plutôt que d'utiliser la méthode `putdata()`.
On pourra utiliser une des deux méthodes suivantes :

creer-image.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  largeur,hauteur = 200,50
6  im1=Image.new('L',(largeur, hauteur))
7  pix1=im1.load()
8  for ligne in range(hauteur):
9      for colonne in range(largeur):
10         pix1[colonne,ligne]=100
11 im1.show()
12
13 im2=Image.new('L',(largeur, hauteur)) # en niveaux de gris
14 im3=Image.new('RGB',(largeur, hauteur)) # en couleurs
15 for ligne in range(hauteur):
16     for colonne in range(largeur):
17         im2.putpixel((colonne,ligne),150)
18         im3.putpixel((colonne,ligne),150)
19 im2.show()
20 im3.show()
```

13.4.2 Réflexions et rotations

Nous avons vu qu'il existait des fonctions prédéfinies dans *PIL* pour faire tourner des images. Nous allons ici le faire en agissant sur les matrices.

Appliquons un effet de miroir horizontal à l'image originale.

Rappelons que le coin supérieur gauche d'une image est repéré par ses coordonnées $(0, 0)$ alors que son coin inférieur droit l'est par $(\text{hauteur} - 1, \text{largeur} - 1)$.

Si l'on souhaite effectuer un effet de réflexion par un miroir horizontal, il faut envoyer le coin supérieur gauche dans le coin inférieur gauche, c'est-à-dire le pixel de coordonnées $(0, 0)$ en $(\text{hauteur} - 1, 0)$. Plus généralement, il faudra envoyer le pixel de coordonnées $(\text{ligne}, \text{colonne})$ en $(\text{hauteur} - 1 - \text{ligne}, \text{colonne})$ (ou l'inverse). Essayons avec le programme ci-dessous :

miroir_horiz.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  # extension de sauvegarde
6  ext='.eps'
7
8  # fonction miroir horizontal
9  def miroir1(image1):
10     # taille de l'image
11     largeur,hauteur = image1.size
12     # création d'une nouvelle image de même taille
13     image2 = Image.new('RGB',(largeur,hauteur))
14     # lecture des données de l'image originale
15     pixels1 = image1.load()
16     # lecture des données de l'image créée
17     pixels2 = image2.load()
18     # on parcourt les pixels (ligne et colonne)
19     for ligne in range(hauteur):
20         for colonne in range(largeur):
21             # on envoie les pixels de l'image1 dans l'image2
22             # en les changeant de place
23             pixels2[colonne,ligne] = pixels1[colonne,hauteur-1-ligne]
24     return image2
25
26 # ouverture de l'image
27 mon_image1 = Image.open("tux20.jpg")
28 # on applique la fonction la photo précédente
29 mirror = miroir1(mon_image1)
30 # on sauve l'image2 créée
31 mirror.save('tux20_mir1'+ext)
32 # on la visualise
33 mirror.show()
```

Le résultat :

(a) tux20 (b) tux20_mir1

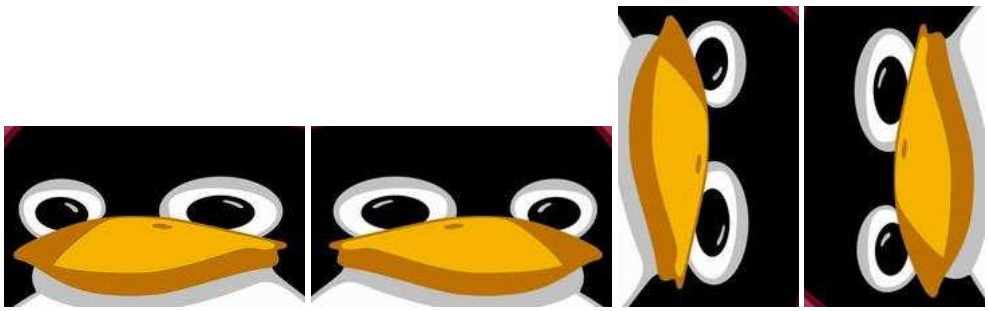
FIGURE 13.10 – Miroir horizontal

Il est très important que la fonction "retourne" une image : en effet, si on souhaite faire subir plusieurs traitements à la suite à l'image de départ, cela ne pourra être possible si la fonction se termine par `show()` ou `save()`. Il est ici de même des fonctions vues auparavant qui devaient "retourner" un résultat et non une instruction `print()`.

⇒ **Activité 13. 55**

Écrivez les 3 autres fonctions réalisant la réflexion par un miroir vertical ainsi que les rotations dans les sens trigonométrique et horaire.

Le résultat :



(a) tux20 (b) tux20_mir2 (c) tux20_rot1 (d) tux20_rot2

FIGURE 13.11 – Rotations

13.4.3 Conversion d'une image en niveaux de gris

Soit le programme suivant :

niveaugris1.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  # ouverture du fichier image source
6  mon_image = Image.open("tux20.jpg")
7
8  # largeur = largeur de l'image (nombre de colonnes)
9  # hauteur = hauteur de l'image (nombre de lignes)
10 largeur,hauteur = mon_image.size
11
12 # création de l'image destination
13 nouvelle_image = Image.new("RGB",(largeur,hauteur))
14
15 # on balaie toutes les lignes de l'image source, de 0 à hauteur-1
16 for ligne in range(hauteur):
17     # pour chaque ligne on balaie toutes les colonnes, de 0 à largeur
                                        -1
18     for colonne in range(largeur):
19         # on stocke le pixel (x,y) dans pix
20         pix = mon_image.getpixel((colonne,ligne))
21         # on calcule le niveau de gris (gris moyen ici)
22         gris = int((pix[0]+pix[1]+pix[2])/3)
23         # pix[0] est la composante rouge
24         # pix[1] la composante verte,
25         # pix[2] la composante bleue
26         # et int() pour avoir un entier compris entre 0 et 255
27
28         # on affecte à chaque couleur la valeur du gris moyen
29         rouge = gris
30         vert = gris
31         bleu = gris
32         # on écrit le pixel modifié sur l'image destination
33         nouvelle_image.putpixel((colonne,ligne),(rouge,vert,bleu))
34
35 # on stocke l'image résultante
36 nouvelle_image.save("tux20_gris_matrice.png")
37 nouvelle_image.save("tux20_gris_matrice.eps")
38
39 # on montre l'image
```

```
40  nouvelle_image.show()
41
42  mon_image = Image.open("tux20_gris_matrice.png")
43  data = list(mon_image.getdata())
44  print(data[-5:-1])
```

La méthode utilisée dans le programme précédent permet de conserver 3 composantes (RGB). On peut le voir en observant le résultat des dernières lignes de code :

```
[(255, 255, 255), (255, 255, 255), (252, 252, 252), (250, 250, 250)]
```

⇒ **Activité 13. 56**

En vous aidant du programme précédent, écrivez-en un autre (*niveaugris2.py*) qui permet de créer réellement une image en niveaux de gris, c'est-à-dire avec une seule bande. Vous pourrez le vérifier en observant le résultat des dernières lignes de code :

```
[255, 255, 252, 250]
```

Le résultat :

(a) tux20_gris_matrice (b) tux20_gris_matrice_bis

FIGURE 13.12 – Niveaux de gris

13.4.4 Négatif en niveaux de gris

Partant de l'image "RGB", on peut la convertir en niveaux de gris puis l'inverser.

⇒ **Activité 13. 57**

Écrivez, à l'aide d'une fonction *complement* que vous créerez, le programme *negatif.py* correspondant. Vous pourrez pour la suite, utiliser la méthode `convert('L')` afin d'être sûr d'avoir une image en niveaux de gris, ceci de manière plus rapide que par le programme précédent, par exemple, comme ceci :

```
image1 = Image.open('tux20.png').convert('L')
```

Le résultat :

(a) tux20 (b) niveaux de gris (c) tux20_inv

FIGURE 13.13 – Inversion des couleurs

13.4.5 Éclaircissement et assombrissement

13.4.5.1 Éclaircissement

Pour éclaircir une image en niveaux de gris, il faut augmenter la valeur des pixels (comprise entre 0 et 255). La plus grande valeur représente le blanc, et la plus petite le noir.

Une solution peut consister à ajouter un certain nombre à la valeur de chaque pixel : c'est ce qui est proposé dans le programme ci-dessous :

eclairciss1.py

```
1 # -*- coding: utf-8 -*-
2
3 from PIL import Image
```

```
4
5  # extension de sauvegarde
6  ext='.eps'
7  nom = 'billes'
8
9  def plus(x,k):
10     if x <= 255-k:
11         return (int(x+k))
12     else:
13         return 255
14
15 def eclair1(image1,k):
16     # taille de l'image
17     largeur,hauteur = image1.size
18     # création d'une nouvelle image de même taille
19     image2 = Image.new('L',(largeur,hauteur))
20     # lecture des données de l'image originale
21     pixels1 = image1.load()
22     # lecture des données de la nouvelle image
23     pixels2 = image2.load()
24     # on parcourt les pixels (ligne et colonne)
25     for ligne in range(hauteur):
26         for colonne in range(largeur):
27             pixels2[colonne,ligne] = plus(pixels1[colonne,ligne],k)
28     return image2
29
30 # ouverture de l'image
31 mon_image1 = Image.open("billes.jpg").convert('L')
32 # on applique la fonction la photo précédente
33 resultat = eclair1(mon_image1,60)
34 # on sauve l'image créée
35 resultat.save(nom+'_eclair1'+ext)
36 # on la visualise
37 resultat.show()
```

Une autre possibilité est de ramener les valeurs de pixels entre 0 et 1, et de leur appliquer une fonction croissante et concave entre 0 et 1 telle que $f(0) = 0$ et $f(1) = 1$. La fonction croissante et concave $f(x) = \sqrt{x}$ convient. Il faut ensuite ramener la valeur des pixels dans l'intervalle $[0, 255]$.

⇒ **Activité 13. 58**

Complétez le programme *eclairciss2.py* suivant pour qu'il fasse l'opération précédemment décrite.

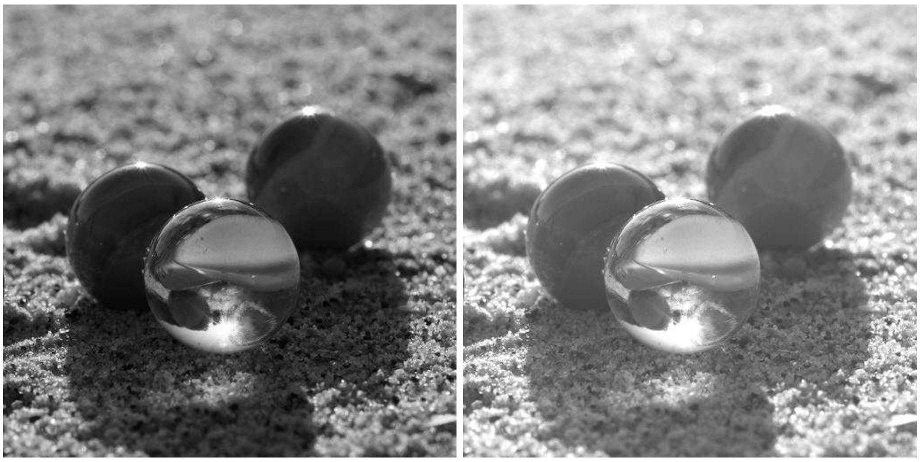
Le programme :

eclairciss2.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4  import numpy as np
5
6  # extension de sauvegarde
7  ext='.eps'
8  nom = 'billes'
9
10 def racine(x):
11
12
13
```

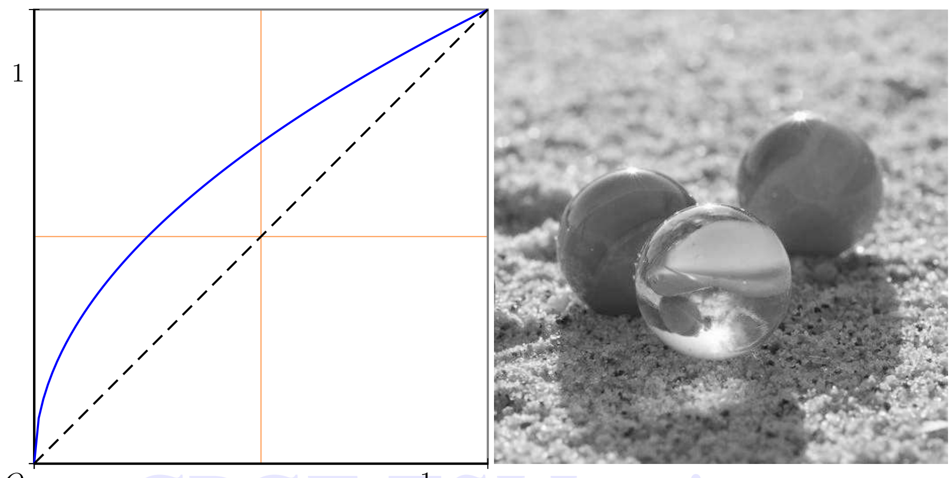
```python
def eclair2(image1):
    # taille de l'image
    largeur,hauteur = image1.size
    # création d'une nouvelle image de même taille
    image2 = Image.new('L',(largeur,hauteur))
    # lecture des données de l'image originale
    pixels1 = image1.load()
    # lecture des données de la nouvelle image
    pixels2 = image2.load()
    # on parcourt les pixels (ligne et colonne)
    for ligne in range(hauteur):
        for colonne in range(largeur):



   return image2

# ouverture de l'image
mon_image1 = Image.open("billes.jpg").convert('L')
# on applique la fonction la photo précédente
resultat = eclair2(mon_image1)
# on sauve l'image créée
resultat.save(nom+'_eclair2'+ext)
# on la visualise
resultat.show()
```

Les résultats :



(a) billes_gris (b) billes_eclair1

FIGURE 13.14 – Éclaircissement 1



(a) $\sqrt{x}$ (b) billes_eclair2

FIGURE 13.15 – Éclaircissement 2

13.4.5.2 Assombrissement

Pour assombrir une image en niveaux de gris, il faut au contraire diminuer la valeur des pixels. Une fonction convient bien : c'est $f(x) = x^2$, qui est croissante et convexe.

⇒ **Activité 13. 59**

Complétez les programmes *assombriss1.py* et *assombriss2.py* sur le même modèle que l'éclaircissement.

On obtient :

assombriss1.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  # extension de sauvegarde
6  ext='.eps'
7  nom = 'billes'
8
9  def moins(x,k):
10     if
11         return
12     else:
13         return
14
15 def assomb1(image1,k):
16     # taille de l'image
17     largeur,hauteur = image1.size
18     # création d'une nouvelle image de même taille
19     image2 = Image.new('L',(largeur,hauteur))
20     # lecture des données de l'image originale
21     pixels1 = image1.load()
22     # lecture des données de la nouvelle image
23     pixels2 = image2.load()
24     # on parcourt les pixels (ligne et colonne)
25     for ligne in range(hauteur):
26         for colonne in range(largeur):
27             pixels2[colonne,ligne] = moins(pixels1[colonne,ligne],k)
28     return image2
29
30 # ouverture de l'image
31 mon_image = Image.open("billes.jpg").convert('L')
32 # on applique la fonction la photo précédente
33 res = assomb1(mon_image,60)
34 # on sauve l'image créée
35 res.save(nom+'_assomb1'+ext)
36 # on la visualise
37 res.show()
```

et :

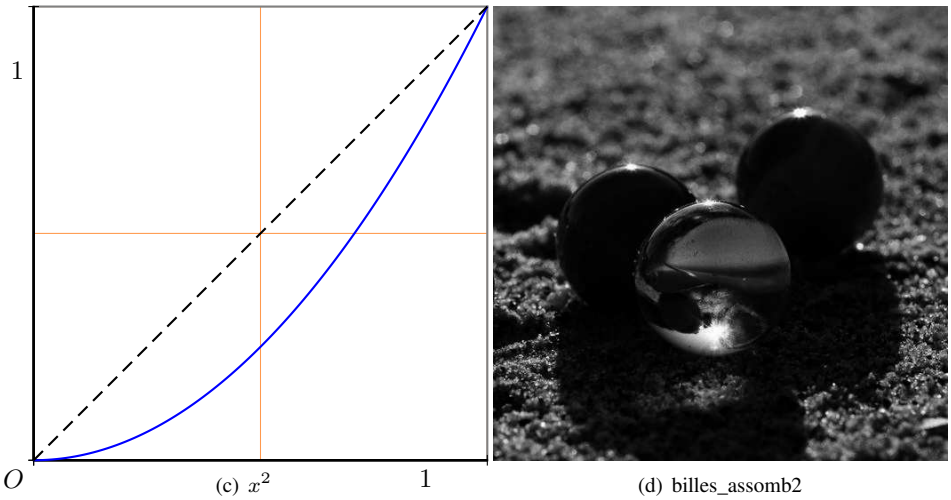
assombriss2.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
```

```
4  import numpy as np
5
6  # extension de sauvegarde
7  ext='.eps'
8  nom = 'billes'
9
10 def carre(x):
11
12
13 def assomb2(image1):
14     # taille de l'image
15     largeur,hauteur = image1.size
16     # création d'une nouvelle image de même taille
17     image2 = Image.new('L',(largeur,hauteur))
18     # lecture des données de l'image originale
19     pixels1 = image1.load()
20     # lecture des données de la nouvelle image
21     pixels2 = image2.load()
22     # on parcourt les pixels (ligne et colonne)
23     for ligne in range(hauteur):
24         for colonne in range(largeur):
25             # on envoie les pixels de l'image1 dans l'image2
26         # en leur appliquant la fonction carré
27         # définie par carre
28         pixels2[colonne,ligne] =
29     return image2
30
31 # ouverture de l'image
32 mon_image1 = Image.open("billes.jpg").convert('L')
33 # on applique la fonction la photo précédente
34 res = assomb2(mon_image1,)
35 # on sauve l'image créée
36 res.save(nom+'_assomb2'+ext)
37 # on la visualise
38 res.show()
```

Les résultats :

(a) billes_gris (b) billes_assomb1



(c) x^2 (d) billes_assomb2

FIGURE 13.16 – Assombrissement

13.4.6 Contraste d'une image

Principe

Pour contraster une image, on applique une fonction telle que :

- $f(0) = 0$
- $f(1) = 1$
- $f(0{,}5) = 0{,}5$

Ainsi, les noirs seront plus noirs et les blancs seront plus blancs.

Bien sûr, ceci est applicable sur des pixels de niveau de gris compris entre 0 et 1. Il faudra donc faire la conversion correspondante.

2 fonctions sont particulièrement adaptées :

- $f(x) = 3x^2 - 2x^3$
- $g(x) = x^3 \left(6x^2 - 15x + 10\right)$

⇒ **Activité 13. 60**

Complétez le programme *contraste.py* pour qu'il réalise ces 2 opérations.

Le programme :

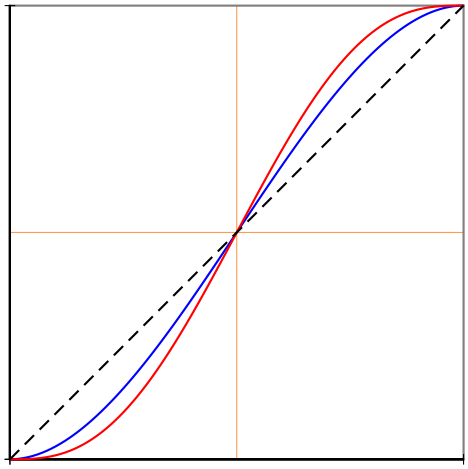
contraste.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4  import numpy as np
5
6  # extension de sauvegarde
7  ext1='.jpg'
8  ext2='.eps'
9  nom = 'billes'
10
11 def f(x):
12
13
14 def g(x):
15
16
17 def cont1(image1):
18     # taille de l'image
19     largeur,hauteur = image1.size
20     # création d'une nouvelle image de même taille
21     image2 = Image.new('L',(largeur,hauteur))
22     # lecture des données de l'image originale
23     pixels1 = image1.load()
24     # lecture des données des nouvelles images
25     pixels2 = image2.load()
26     # on parcourt les pixels (ligne et colonne)
27     for ligne in range(hauteur):
28         for colonne in range(largeur):
29             pixels2[colonne,ligne] =
30     return image2
31
32 # ouverture de l'image
```

```
33  mon_image1 = Image.open("billes.jpg").convert('L')
34  # on applique la fonction cont1 à la photo précédente
35  resultat = cont1(mon_image1)
36  # on sauve les images créées
37  resultat.save(nom+'_contr1'+ext1)
38  resultat.save(nom+'_contr1'+ext2)
39  # on visualise
40  Image.open('billes_contr1.jpg').show()
41
42  def cont2(image1):
43      # taille de l'image
44      largeur,hauteur = image1.size
45      # création d'une nouvelle image de même taille
46      image2 = Image.new('L',(largeur,hauteur))
47      # lecture des données de l'image originale
48      pixels1 = image1.load()
49      # lecture des données des nouvelles images
50      pixels2 = image2.load()
51      # on parcourt les pixels (ligne et colonne)
52      for ligne in range(hauteur):
53          for colonne in range(largeur):
54              pixels2[colonne,ligne] =
55      return image2
56
57  # ouverture de l'image
58  mon_image1 = Image.open("billes.jpg").convert('L')
59  # on applique la fonction cont2 à la photo précédente
60  resultat = cont2(mon_image1)
61  # on sauve les images créées
62  resultat.save(nom+'_contr2'+ext1)
63  resultat.save(nom+'_contr2'+ext2)
64  # on visualise
65  Image.open('billes_contr2.jpg').show()
```

Les résultats :

(a) billes_gris



(b) $f(x)$, $g(x)$

(c) billes_contr1 (d) billes_contr2

FIGURE 13.17 – Contraste

Remarque

On peut aussi composer les fonctions précédentes pour améliorer le contraste d'une image. Exemple avec le fichier *contraste_comp.py* suivant :

contraste_comp.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  # extension de sauvegarde
6  ext1='.png'
7  ext2='.eps'
8  nom = 'billes'
9
10 def f(x):
11
12
13 def g(x):
14
15
16 # composition des fonctions : f o g
17 def cont3(image1):
18     # taille de l'image
19     largeur,hauteur = image1.size
20     # création d'une nouvelle image de même taille
21     image2 = Image.new('L',(largeur,hauteur))
22     # lecture des données de l'image originale
23     pixels1 = image1.load()
24     # lecture des données des nouvelles images
25     pixels2 = image2.load()
26     # on parcourt les pixels (ligne et colonne)
27     for ligne in range(hauteur):
28         for colonne in range(largeur):
29             pixels2[colonne,ligne] =\
30                 int(f(g(pixels1[colonne,ligne]/255))*255)
31     return image2
32
33 # ouverture de l'image
34 mon_image1 = Image.open(nom+".jpg").convert('L')
35 # on applique la fonction cont3 à la photo précédente
36 resultat = cont3(mon_image1)
```

```
37 # on sauve les images créées
38 resultat.save(nom+'_contr3'+ext2)
39 resultat.save(nom+'_contr3'+ext1)
40 Image.open(nom+'_contr3.png').show()
41
42 # ou bien plus simplement :
43 # resultat_bis = cont1(cont2(mon_image1))
```

13.4.7 Transparence d'une image

13.4.8 La transparence

Les formats *.gif* et *.png* offrent la possibilité de rendre le fond de l'image transparent. Ceci permet de poser l'image sur des fonds de couleur, sans que l'on voit par exemple le blanc de l'image qui constitue son propre fond.

Le mode "RGBA" d'une image permet de gérer cette transparence. C'est le dernier canal (alpha) qui traduit celle-ci : 255 pour opaque, 0 pour transparent.

⇒ **Activité 13. 61**

Complétez le programme *transpar.py* suivant afin que la transparence passe de 255 à 100 :

Le programme :

transpar.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  mon_image = Image.open("billes.jpg")
6  largeur, hauteur = mon_image.size
7
8  # conversion de l'image RGB en RGBA (A pour alpha, la transparence)
9  mon_image_bis = mon_image.convert("RGBA")
10 nouvelle_image = Image.new("RGBA",(largeur,hauteur))
11
12 for ligne in range(hauteur):
13     for colonne in range(largeur):
14         pixel = mon_image_bis.getpixel((colonne,ligne))
15         rouge =
16         vert =
17         bleu =
18         alpha =
19         nouvelle_image.putpixel(                                    )
20
21 nouvelle_image.save("billes_transp.png","PNG")
22 nouvelle_image.show()
```

13.4.9 Seuillage d'une image

Segmentation

La segmentation consiste à découper une image en régions distinctes. Cette segmentation peut se réaliser par deux méthodes :

- par contours
- par homogénéité, par exemple en séparant les zones de même couleur

Le seuillage d'image est la méthode la plus simple de segmentation d'image. À partir d'une image en niveau de gris, le seuillage d'image peut être utilisé pour créer une image comportant uniquement deux valeurs, noir ou blanc : l'image est alors en noir et blanc (monochrome).

Principe

Le seuillage d'image remplace un à un les pixels d'une image à l'aide d'une valeur seuil fixée (par exemple 122). Ainsi, si un pixel à une valeur supérieure au seuil (par exemple 150), il prendra la valeur 255 (blanc), et si sa valeur est inférieure (par exemple 100), il prendra la valeur 0 (noir).

13.4.9.1 Seuillage à 1 seuil d'une image en niveaux de gris

Le programme *seuillage.py* réalise un seuillage à un seuil, donc une binarisation :

seuillage.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  def seuil(x):
6      if x<122:
7          return 0
8      else:
9          return 255
10
11 # seuillage d'une image en niveaux de gris
12 mon_image = Image.open('tux20.jpg').convert('L')
13 pixel = mon_image.load()
14 for colonne in range(mon_image.size[0]):
15     for ligne in range(mon_image.size[1]):
16         pixel[colonne,ligne] = seuil(pixel[colonne,ligne])
17
18 mon_image.save('tux20_seuil.eps')
19 mon_image.show()
```

Le résultat :

FIGURE 13.18 – tux20_seuil

13.4.9.2 Seuillage à 2 seuils d'une image en niveaux de gris

On peut réaliser l'opération avec 2 seuils : par exemple, tous les pixels de valeur inférieure à 100 sont mis à 0 et tous ceux de valeur supérieure à 150 sont mis à 255. On obtient alors du blanc, du noir et une teinte de gris.

⇒ **Activité 13. 62**

Modifiez le programme précédent et l'enregistrer sous le nom de *seuillage_2seuils.py* de façon à ce qu'il permette de réaliser cette opération.

Le programme :

Le résultat :

FIGURE 13.19 – tux20_seuil2

13.4.9.3 Image en couleurs

On peut étendre le principe avec une image en couleurs. On peut regarder le programme *seuillageRGB.py* :

seuillageRGB.py

```
1 # -*- coding: utf-8 -*-
2
3 from PIL import Image
```

```
4
5  def seuil(x):
6      if x<180:
7          return 0
8      else:
9          return 255
10
11 fichier='tux20'
12 # Essai avec billes :
13 # fichier='billes'
14
15 ext='.jpg'
16 fich_ext=fichier+ext
17
18 # Seuillage d'une image en couleurs
19 mon_image = Image.open(fich_ext)
20 composition = mon_image.split()
21 largeur,hauteur = mon_image.size
22
23 rouge = list(composition[0].getdata())
24 vert = list(composition[1].getdata())
25 bleu = list(composition[2].getdata())
26
27 # On recrée l'image rouge
28 for i in range(len(rouge)):
29     rouge[i] = seuil(rouge[i])
30 nouveau_rouge = Image.new("L",(largeur,hauteur))
31 nouveau_rouge.putdata(rouge)
32
33 # On recrée l'image verte
34 for i in range(len(vert)):
35     vert[i] = seuil(vert[i])
36 nouveau_vert = Image.new("L",(largeur,hauteur))
37 nouveau_vert.putdata(vert)
38
39 # On recrée l'image bleue
40 for i in range(len(bleu)):
41     bleu[i] = seuil(bleu[i])
42 nouveau_bleu = Image.new("L",(largeur,hauteur))
43 nouveau_bleu.putdata(bleu)
44
45 # Composition de l'image
46 ma_composition=nouveau_rouge,nouveau_vert,nouveau_bleu
47
48 # Recomposition de l'image
49 mon_image_recomposee = Image.merge('RGB',ma_composition)
50
51 # Sauvegarde de l'image décomposée puis recomposée
52 mon_image_recomposee.save(fichier+"_seuilRGB.eps")
53 mon_image_recomposee.save(fichier+"_seuilRGB.png")
54
55 # Visualisation de l'image recomposée
56 mon_image_recomposee.show()
```

Le résultat :

FIGURE 13.20 – tux20_seuilRGB

Remarque

Si vous voulez tester l'image *billes.jpg*, il vous suffit de commenter la ligne 12 et de décommenter la ligne 14.

13.4.10 Filtrage

Nous nous limiterons dans cette partie à filtrer des images en niveaux de gris mais ceci est réalisable bien sûr sur chacune des composantes d'une image en couleurs.

13.4.10.1 Filtrage du contour

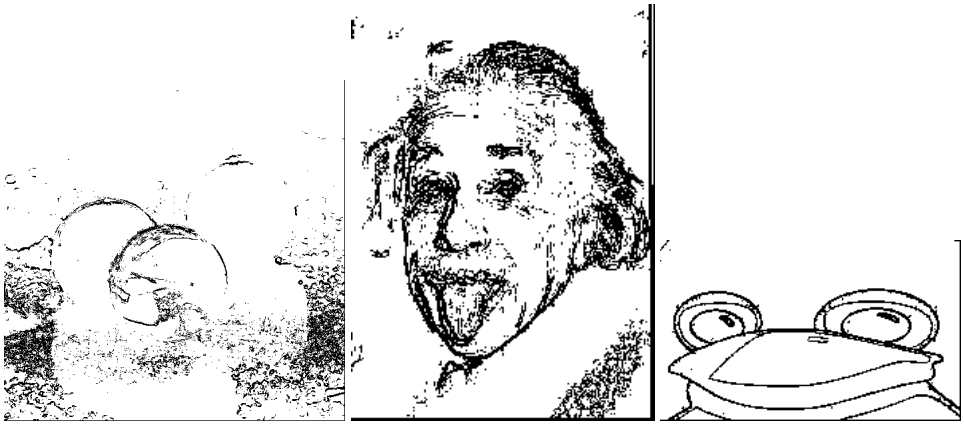
Filtrer le contour d'une image se fait à partir de l'image en niveaux de gris : il vaut mieux donc la convertir tout d'abord. Pour filtrer les contours, on peut noircir les pixels dont la différence de valeurs avec leurs voisins excède un certain seuil (souvent inférieur à 25). C'est ce que réalise le programme *contour1.py* :

contour1.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  ext= '.eps'
6
7  mon_image1 = Image.open("billes.jpg")
8  mon_image2 = Image.open('einstein1.png')
9  mon_image3 = Image.open('tux20_gris.png')
10
11 def filtre_contour(image1,seuil):
12     largeur,hauteur = image1.size
13     image2 = Image.new('L',(largeur,hauteur))
14     image1 = image1.convert('L')
15     pixels1 = image1.load()
16     pixels2 = image2.load()
17     for ligne in range(hauteur-1):
18         for colonne in range(largeur-1):
19             if abs(pixels1[colonne,ligne]-pixels1[colonne,ligne+1])\
20                   > seuil or\
21                   abs(pixels1[colonne,ligne]-pixels1[colonne+1,ligne]
                                                  ) > seuil:
22                 pixels2[colonne,ligne] = 0
23             else:
24                 pixels2[colonne,ligne] = 255
25     return image2
26
27 res1 = filtre_contour(mon_image1,20)
28 res1.save('billes_cont'+ext)
29 res1.show()
30 res2 = filtre_contour(mon_image2,20)
```

```
31  res2.save('einstein1_cont'+ext)
32  res2.show()
33  res3 = filtre_contour(mon_image3,20)
34  res3.save('tux20_gris_cont'+ext)
35  res3.show()
```

Le résultat :



(a) billes_cont (b) einstein11_cont (c) tux20_contour

FIGURE 13.21 – Filtrage du contour1

Le filtrage peut se faire à l'aide d'un filtre à matrice de convolution. Le principe est assez simple : on "fait passer" un noyau (en pratique, une matrice 2 x 2, 3 x 3, ...) sur la matrice de l'image à traiter. Exemple de masque 3 x 3 :

a	b	c
d	e	f
g	h	i

Soit la matrice image ci-contre. Supposons qu'elle soit en niveaux de gris.

53	47	72	98	118	136	165
58	37	39	84	130	130	167
63	34	28	80	101	139	180
69	23	29	73	112	140	174
64	21	20	67	127	145	160
61	18	12	79	104	113	121
12	15	19	66	102	108	119

Principe

L'application du filtre va consister à remplacer la case centrale colorée en rouge de valeur 80 par la valeur s_t donnée par la somme des 9 produits :
$s_t = 39\,a + 84\,b + 130\,c + 28\,d + 80\,e + 101\,f + 29\,g + 73\,h + 112\,i$

Le problèmes suivants vont être à considérer :

1. Sur un pixel appartenant à la bordure de l'image, il manque des voisins,
2. Le résultat peut sortir de l'intervalle $[0, 255]$ (c'est même probable !)

Les solutions possibles sont les suivantes :

1. (a) On ignore les pixels des bords et on parcourt donc la matrice image à traiter de la deuxième ligne à l'avant-dernière ligne et de la deuxième colonne à l'avant-dernière colonne.
   (b) On crée une bordure (par exemple noire ou blanche), par exemple de 1 pixel de chaque côté et on reprend la méthode précédente.
2. (a) Chaque valeur de pixel est divisée par la somme s_c des cœfficients de la matrice *noyau* : on parle alors de normalisation. Si cette somme s_c est nulle, on divise par 1. Dans ce dernier cas, on ajoute 128 pour éviter les valeurs négatives.
   (b) Chaque valeur de pixel est écrêtée par 0 ou 255.
   (c) L'ensemble de la matrice résultat est ramenée dans l'intervalle $[0, 255]$ à l'aide d'une règle de 3.

On peut ainsi écrire les programmes *contour2.py* :

contour2.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  ext1='.jpg'
6  ext2='.eps'
7
8  # détecteurs de contours
9  noyau1 =\
10 [-1,-1,-1,\
11 -1 , 8,-1,\
12 -1 ,-1,-1]
13
14 def filtre_contour1(image1,noyau):
15     largeur,hauteur = image1.size
16
17     # conversion en niveaux de gris, mode RGB
18     image1 = image1.convert("L")
19     image2 = Image.new("L",(largeur,hauteur))
20     # filtrage de l'image en niveaux de gris
21     for y in range(1,hauteur-1):
22         for x in range(1,largeur-1):
23             liste_pix=[]
24             liste_pix.append(image1.getpixel((x-1,y-1)))
25             liste_pix.append(image1.getpixel((x,y-1)))
26             liste_pix.append(image1.getpixel((x+1,y-1)))
27             liste_pix.append(image1.getpixel((x-1,y)))
28             liste_pix.append(image1.getpixel((x,y)))
29             liste_pix.append(image1.getpixel((x+1,y)))
30             liste_pix.append(image1.getpixel((x-1,y+1)))
31             liste_pix.append(image1.getpixel((x,y+1)))
32             liste_pix.append(image1.getpixel((x+1,y+1)))
33             pixel=0
34             for i in range(9):
35                 pixel=int(pixel+noyau[i]*liste_pix[i])
36
37             pixel = pixel + 128
38             image2.putpixel((x,y),pixel)
39     return image2
40
41 nom_image = "billes"
42 mon_image = Image.open(nom_image+ext1)
43 filtre1 = filtre_contour1(mon_image,noyau1)
44 filtre1.save(nom_image+'_contours11'+ext2)
45 filtre1.show()
```

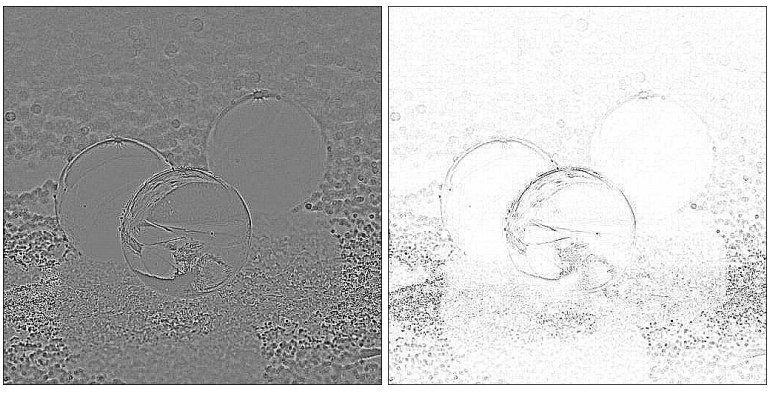
et *filtre_contour3* :

filtre_contour3

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  ext1='.jpg'
6  ext2='.eps'
7
8  # détecteurs de contours
```

```
noyau1 =\
[-1,-1,-1,\
-1 , 8,-1,\
-1 ,-1,-1]

def filtre_contour2(image1,noyau):
    largeur,hauteur = image1.size

    # conversion en niveaux de gris, mode RGB
    image1 = image1.convert("L")
    image2 = Image.new("L",(largeur,hauteur))
    # filtrage de l'image en niveaux de gris
    for y in range(1,hauteur-1):
        for x in range(1,largeur-1):
            liste_pix=[]
            liste_pix.append(image1.getpixel((x-1,y-1)))
            liste_pix.append(image1.getpixel((x,y-1)))
            liste_pix.append(image1.getpixel((x+1,y-1)))
            liste_pix.append(image1.getpixel((x-1,y)))
            liste_pix.append(image1.getpixel((x,y)))
            liste_pix.append(image1.getpixel((x+1,y)))
            liste_pix.append(image1.getpixel((x-1,y+1)))
            liste_pix.append(image1.getpixel((x,y+1)))
            liste_pix.append(image1.getpixel((x+1,y+1)))
            pixel=0
            for i in range(9):
                pixel=int(pixel+noyau[i]*liste_pix[i])

            if pixel < 0:
                pixel =0
            if pixel > 255:
                pixel = 255
            pixel = pixel + 128
            image2.putpixel((x,y),pixel)
    return image2

nom_image = "billes"
mon_image = Image.open(nom_image+ext1)
filtre1 = filtre_contour2(mon_image,noyau1)
filtre1.save(nom_image+'_contours21'+ext2)
filtre1.show()
```

qui donnent :



(a) billes_contours11 (b) billes_contours21

FIGURE 13.22 – Filtrage du contour

13.4.10.2 Embossage

En utilisant la méthode précédente, ce filtrage sert à mettre en relief l'image. Le noyau est une matrice 3 x 3 et le décalage est de 128. Voici le noyau utilisé :

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Le programme :

embossage.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  ext1='.jpg'
6  ext2='.eps'
7
8  noyau0 =\
9  [-2 , -1, 0,\
10 -1  , 0, 1,\
11 0 , 1, 2]
12
13 def embossage(image1,noyau):
14     image2 = Image.new("RGB",image1.size)
15     image3 = Image.new("RGB",image1.size)
16
17     # conversion en niveaux de gris, mode RGB
18     for ligne in range(image1.size[1]):
19          for colonne in range(image1.size[0]):
20              pix = image1.getpixel((colonne,ligne))
21              rouge = int((pix[0]+pix[1]+pix[2])/3)
22              vert = rouge
23              bleu = rouge
24              image2.putpixel((colonne,ligne),(rouge,vert,bleu))
25
26     # filtrage de l'image en niveaux de gris
27     for ligne in range(1,image1.size[1]-1):
28         for colonne in range(1,image1.size[0]-1):
29             liste_pix=[]
30             liste_pix.append(image2.getpixel((colonne-1,ligne-1)))
31             liste_pix.append(image2.getpixel((colonne,ligne-1)))
32             liste_pix.append(image2.getpixel((colonne+1,ligne-1)))
33             liste_pix.append(image2.getpixel((colonne-1,ligne)))
34             liste_pix.append(image2.getpixel((colonne,ligne)))
35             liste_pix.append(image2.getpixel((colonne+1,ligne)))
36             liste_pix.append(image2.getpixel((colonne-1,ligne+1)))
37             liste_pix.append(image2.getpixel((colonne,ligne+1)))
38             liste_pix.append(image2.getpixel((colonne+1,ligne+1)))
39             rouge=0
40             for i in range(9):
41                 rouge=int(rouge+noyau[i]*liste_pix[i][0])
42
43             rouge = rouge+128
44             vert = rouge
45             bleu = rouge
46             image3.putpixel((colonne,ligne),(rouge,vert,bleu))
```

```
47
48      return image3
49
50  nom_image = "billes"
51  mon_image = Image.open(nom_image+ext1)
52  emboss = embossage(mon_image,noyau0)
53  emboss.save(nom_image+'_relief'+ext2)
54  emboss.save(nom_image+'_relief'+ext1)
55  emboss.show()
```

Le résultat :

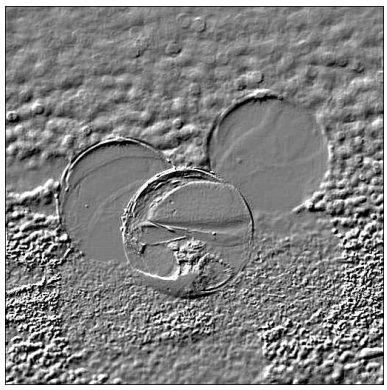


FIGURE 13.23 – billes_relief

On peut par exemple s'amuser à embosser l'image `quimper-cathedrale.jpg`.

13.4.10.3 Flou gaussien

Ce filtrage sert à atténuer les changements brusques d'intensité. On utilise la méthode précédente. La matrice est une matrice 5 x 5 et la normalisation est égale à 273. Voici le noyau utilisé :

$$\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Le programme :

flou-gaussien.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  ext1='.jpg'
6  ext2='.eps'
7
8  noyau0 =\
9  [1,  4,   7,   4,  1,\
10 4,  16,  26,  16,  4,\
11 7,  26,  41,  26,  7,\
12 4,  16,  26,  16,  4,\
```

```
13  1,   4,   7,   4,  1 ]
14
15  def flou_gaussien(image1,noyau):
16      image2 = Image.new("RGB",image1.size)
17      image3 = Image.new("RGB",image1.size)
18
19      # conversion en niveaux de gris, mode RGB
20      for ligne in range(image1.size[1]):
21           for colonne in range(image1.size[0]):
22               pix = image1.getpixel((colonne,ligne))
23               rouge = int((pix[0]+pix[1]+pix[2])/3)
24               vert = rouge
25               bleu = rouge
26               image2.putpixel((colonne,ligne),(rouge,vert,bleu))
27
28      # filtrage de l'image en niveaux de gris
29      for ligne in range(2,image1.size[1]-2):
30          for colonne in range(2,image1.size[0]-2):
31              liste_pix=[]
32              for x in range(colonne-2,colonne+3):
33                  for y in range(ligne-2,ligne+3):
34                      liste_pix.append(image2.getpixel((x,y)))
35              rouge=0
36              for i in range(25):
37                  rouge=rouge+noyau[i]*liste_pix[i][0]
38
39              rouge = int(rouge/273)
40              vert = int(rouge)
41              bleu = int(rouge)
42              image3.putpixel((colonne,ligne),(rouge,vert,bleu))
43
44      return image3
45
46  nom_image = "billes"
47  mon_image = Image.open(nom_image+ext1)
48  emboss = flou_gaussien(mon_image,noyau0)
49  emboss.save(nom_image+'_gauss'+ext2)
50  emboss.save(nom_image+'_gauss'+ext1)
51  emboss.show()
```

Le résultat :

FIGURE 13.24 – billes_gauss

13.4.10.4 Autres filtres

Il existe une multitude de filtres (passe-bas, passe-haut, réhausseur de contraste, ...). Si vous souhaitez utiliser des matrices de convolutions, en voici quelques-unes :

- $\begin{bmatrix} -\frac{1}{6} & -\frac{2}{3} & -\frac{1}{6} \\ -\frac{2}{3} & \frac{13}{3} & -\frac{2}{3} \\ -\frac{1}{6} & -\frac{2}{3} & -\frac{1}{6} \end{bmatrix}$
- $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$
- $\begin{bmatrix} 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 \\ 4 & 4 & 0 & 2 & 2 \\ 4 & 4 & 3 & 3 & 3 \\ 4 & 4 & 3 & 3 & 3 \end{bmatrix}$
- $\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$
- filtre binomial (normalisation à 256)

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- filtre pyramidal (normalisation à 81)

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

- filtre pyramidal (normalisation à 25)

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 2 & 0 \\ 1 & 2 & 5 & 2 & 1 \\ 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
- $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
- $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$
- $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$
- $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$
- $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & 1 \end{bmatrix}$
- $\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$
- $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
- ...

Il est également possible de combiner ces différents traitements.

13.4.11 Histogrammes

Un premier programme que vous devez absolument comprendre :

histo10-ss-csv.py

```
1  #-*- coding: utf-8 -*-
2
3  from PIL import Image
4  import matplotlib.pyplot as plt
5
6  mon_image = Image.open("billes.jpg")
7  hist = [] # ou hist = list()
8
9  largeur,hauteur = mon_image.size
10
11 for i in range(256):
12     hist.append(0)
13 for ligne in range(hauteur):
14     for colonne in range(largeur):
15         p = mon_image.getpixel((colonne,ligne))
16         n_gris = int((p[0]+p[1]+p[2])/3)
17         hist[n_gris] = hist[n_gris]+1
18
19 n_abscisses = list(range(256))
20 plt.xlim(0,255)
21 plt.plot(n_abscisses,hist)
22 plt.grid(True)
23 plt.show()
```

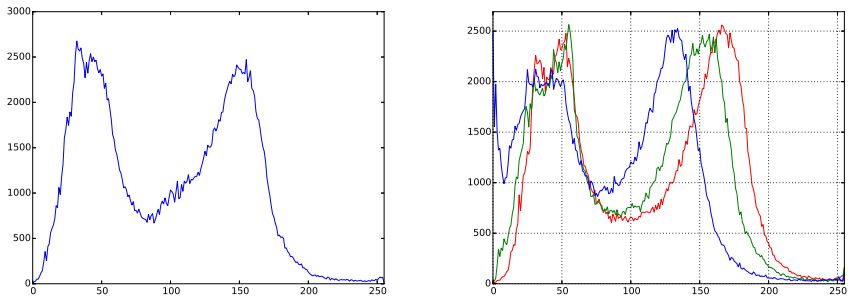
Le même avec écriture dans un fichier `.csv` :

histo10.py

```
1  #-*- coding: utf-8 -*-
2
3  from PIL import Image
4  import csv
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  mon_image = Image.open("billes.jpg")
9  hist = [] # ou hist = list()
10
11 with open("billes_histo_niv_gris.csv","w") as sortie:
12     largeur,hauteur = mon_image.size
13     mon_fichier = csv.writer(sortie,delimiter=';')
14     mon_fichier.writerow(["niveau_gris","nb_pixels"]) # titres
15
16     for i in range(256):
17         hist.append(0)
18     for ligne in range(hauteur):
19         for colonne in range(largeur):
20             p = mon_image.getpixel((colonne,ligne))
21             n_gris = int((p[0]+p[1]+p[2])/3)
22             hist[n_gris] = hist[n_gris]+1
23
24     for i in range(256):
25         mon_fichier.writerow([i,hist[i]])
26
27 with open("billes_histo_niv_gris.csv", "r") as fich:
```

```
28      fichier = csv.reader(fich,delimiter=';')
29      abscisses = []
30      ordonnees = []
31      for ligne in fichier:
32          if ligne==[]:
33              pass
34          else:
35              try:
36                  abscisses.append(float("".join(ligne[0].split(":"))))
37                  ordonnees.append(float("".join(ligne[1].split(":"))))
38              except ValueError: # à cause des titres
39                  pass
40
41  plt.xlim(0,255)
42  plt.plot(abscisses,ordonnees)
43  plt.savefig("histo10.eps")
44  plt.grid(True)
45  plt.show()
```

Les résultats :



(a) histogramme personnel (b) histogramme couleurs + image

FIGURE 13.25 – Histogrammes

⇒ **Activité 13. 63**

Comme vous pouvez le voir ; l'histogramme de droite porte sur les 3 composantes : à vous de le faire dans le programme *histo10-ss-csv-coul.py*.

Le programme :

13.5 Mini-projets sur les images

13.5.1 Conversion d'images

L'objectif de ce projet est d'écrire un programme de conversion d'image.

Celui-ci devra proposer un menu à l'utilisateur. Ce dernier aura le choix dans un premier temps entre :

- binarisation (conversion en noir et blanc),
- conversion en niveaux de gris,
- conversion en négatif de l'image en niveaux de gris,
- conversion en sépia.
- conversion en négatif de l'image en couleurs.

Cette liste n'est pas exhaustive et pourra bien sûr être complétée avec d'autres transformations (vues dans le cours, dans les activités, les exercices, ...).

Pour réaliser le script, on sera limité dans l'utilisation du module *PIL*. On ne pourra utiliser dans celui-ci que les instructions suivantes :

```
mon_image = Image.open(nom_fichier)
mon_image.size
image.getpixel((x,y))
image.putpixel((x,y),(r,v,b))
```


Il faudra donc parcourir les matrices des images.

Le menu pourra s'inspirer du programme *menu_vide.py* :

menu_vide.py

```
# -*- coding: utf-8 -*-

from PIL import Image

def.........

if __name__ == "__main__":
    nb_transfos = ...........
    choix=-1
    while choix !=0:
        print ("Quelle opération désirez-vous faire ?")
        print ("0 - Pour sortir")
        print ("1 ................")
        ............................
        ............................
        while True:
            try:
                choix = int(input())
            except ValueError:
                print("Vous n'avez pas entré un nombre entre 0 et %s
                                                !"\
                %nb_transfos)
                continue
            if choix>=0 and choix<=nb_transfos:
                break
        if choix == 0:
            break
        while True:
            try:
                nom_fichier = input("Entrez le nom de votre fichier
                                                image : ")
                mon_image = Image.open(nom_fichier)
                largeur,hauteur = mon_image.size
            except Exception:
                print("Le fichier", nom_fichier,"est introuvable")
                continue
            break
        if choix == 1:
            .............
        mon_image.show()
```

Remarque

- Pour convertir une image en sepia, **il faut d'abord passer en niveaux de gris**, mais en mode "RGB". Ensuite, on applique les règles suivantes :
  - Si le rouge est inférieur ou égal à 62, on multiplie le rouge par 1,1 et le bleu par 0,9,
  - Si le rouge est compris entre 63 et 191, on multiplie le rouge par 1,15 et le bleu par 0,85,
  - Si le rouge est supérieur à 192, on le remplace par le minimum entre 255 et le rouge multiplié par 1,08. Quant au bleu, on le multiplie par 0,93. La fonction `min()` existe sous *Python*.
- Comme déjà vu, créer le négatif d'une image consiste à inverser toutes les couleurs : le 0 devient 255 et vice-versa.

⇒ **Activité 13. 64**

- Écrivez la fonction permettant de réaliser la conversion en sepia.
- Complétez le menu fourni pour que le programme fasse les conversions demandées.

Vous enregistrerez le programme sous le nom *menu_total.py*.

Le programme :

13.5.2 Stéganographie

La stéganographie consiste à cacher un message ou une image dans une autre image.

13.5.2.1 Message caché dans une image

On va ici cacher une chaîne de caractères dans la photo *einstein1.png*.

La chaîne à crypter est par exemple : "Les sciences, c'est de la balle !" Si vous voulez changer, c'est possible mais plus il est long, plus l'image devra être grande.

Pour cela on va créer une image avec Python qui contiendra le message. Elle s'appellera *einstein11.png*.

Principe

- On va travailler sur la composante rouge des images.
  On rappelle que chaque pixel est défini par un triplet de trois nombres entiers compris entre 0 et 255, le premier donnant la composante rouge, le deuxième la verte et le troisième la bleue.
- Dans le tableau ci-dessous, on donne à la deuxième ligne la composante rouge des 16 premiers pixels.
  Dans une première étape, à la ligne 3 du tableau, les valeurs sont réduites au plus grand nombre pair inférieur ou égal à la valeur.
- On va ajouter ensuite à ces nombres pairs des 0 et des 1 et ce sont ces 0 et ces 1, regroupés par 8, qui vont transmettre les informations cachées.
- La chaîne contient 33 caractères. Dans les huit premiers pixels, on entre l'information 00100001 qui est la représentation binaire de 33.
  On va donc coder l'image sur $8 + 8 * 33 = 272$ pixels.
- Dans les huit pixels suivants, on rentre l'information 01001100 qui est la représentation binaire de 76 et qui est le code *ascii* de L (l majuscule).
  On a donc récupéré le premier caractère de la chaîne.
- La quatrième ligne contient donc la longueur de la chaîne et le premier caractère

pixel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
einstein1	61	59	56	53	52	54	56	57	62	62	63	63	64	65	65	66
réduction	60	58	56	52	52	54	56	56	62	62	62	62	64	64	64	66
chaîne	0	0	1	0	0	0	0	1	0	1	0	0	1	1	0	0
einstein11	60	58	57	52	52	54	56	57	62	63	62	62	65	65	64	66

TABLE 13.4 – message caché

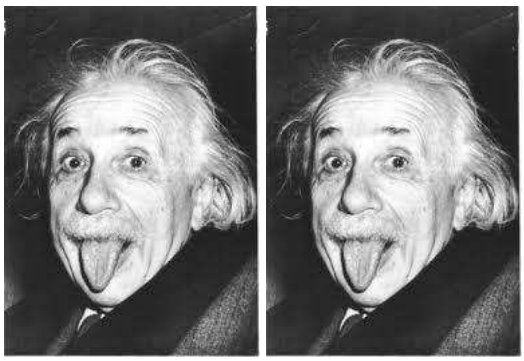
Le programme *message.py* réalise cette série d'opérations (la fonction `lireTexte(nomFichier)` permet de lire un texte enregistré dans fichier *nom.Fichier.txt* s'il est dans le même dossier) :

message.py

```
1  # -*- coding: utf-8 -*-
2
3  def lireTexte(nomFichier) :
4      try:
5          fichier = open(nomFichier, 'r',encoding="utf-8")
6      except IOError:
7          print (nomFichier + " : ce fichier n'existe pas")
8          return None
9      texte = ''
10     while True:
11         ligne = fichier.readline()
12         if ligne == '':
13             break
```

```
        for mot in ligne:
            texte = texte + mot
    fichier.close()
    return texte

from PIL import Image

mon_image = Image.open("einstein1.png")
mon_image.show()
# pour le prof, sauve au format .eps
mon_image.save("einstein1.eps")
# on éclate l'image en trois (rouge vert bleu)
red,green,blue=mon_image.split()
# on transforme le rouge de l'image en liste
rouge=list(red.getdata())

# message à insérer
mon_message = lireTexte('message.txt')

# on note la longueur de la chaine et on la transforme en binaire
# on enlève le "0b" devant avec [2:] et on ajuste à 8 bits
longueur=len(mon_message)
longueur_bin=bin(len(mon_message))[2:].rjust(8,"0")

# la longueur de la chaîne vaut ici 33 qui en base 2 donne 100001

# on transforme la chaine en une liste de 0 et de 1
long=[bin(ord(caractere))[2:].rjust(8,"0") for caractere in
                                        mon_message]

# transformation de la liste en chaine
chaine=''.join(long)

# on code la longueur de la liste dans les 8 premiers pixels rouges
# après pairisation
for j in range(8):
    rouge[j]=2*(rouge[j]//2)+int(longueur_bin[j])
# on code la chaine dans les pixels suivants après pairisation
for i in range(8*longueur):
    rouge[i+8]=2*(rouge[i+8]//2)+int(chaine[i])

# on recrée l'image rouge
nouveau_rouge = Image.new("L",(largeur,hauteur))
nouveau_rouge.putdata(rouge)
# fusion des trois nouvelles images
nouvelle_image = Image.merge('RGB',(nouveau_rouge,green,blue))
nouvelle_image.save("einstein1-code.png")
nouvelle_image.show()
nouvelle_image.save("einstein1-code.eps")
```

Le résultat :



(a) einstein1 (b) einstein1-code

FIGURE 13.26 – message caché

⇒ **Activité 13. 65**

Écrivez le programme en *Python* de façon à découvrir le message caché dans l'image *einstein1-code.png*.
Vous le nommerez par exemple *decodage.py*.
Vous aurez besoin ici de la fonction `str()` qui sert à convertir une donnée en chaîne.
L'instruction `int(ch,2)` permet de convertir un binaire `ch` sous forme de chaîne en nombre décimal.
L'instruction chr(nombre) permet de traduire l'entier `nombre` sous forme d'entier en caractère codé en *ascii*.

```
>>> str(5)
'5
>>> int("01101001",2)
105
>>> chr(105)
"i"
```

13.5.2.2 Image cachée dans une autre

Pour cacher une image dans une autre, nous prendrons 2 images de même dimension et au format *.tif* de façon à ne pas perdre d'information en route.

Principe

Dans l'écriture d'un nombre, qu'il soit décimal ou binaire, les termes de gauche possèdent un poids plus important. Par exemple, pour $57 = 5 \text{ x } 10^1 + 7 \text{ x } 10^0$, le 5 représente 50, plus proche de la valeur réelle du nombre initial. Il en est de même pour sa version binaire 111001 où le 1 de gauche représente $1 \text{ x } 2^5$.
Les couleurs étant codées sur 8 bits, on ne va retenir que les 4 premiers bits, de poids le plus important (les bits "forts"). Exemple avec le programme suivant :

```
16            image2.putpixel((colonne,ligne),(rouge,vert,bleu))
17
18    image2.save('billes_allege.tif')
19    image2.save('billes_allege.eps')
```

On peut ensuite comparer les 2 images :

```
21    image2.sh
```


(a) billes.tif (b) billes_allege.tif

FIGURE 13.27 – stegano1

Considérons le dernier pixel de l'*image10* codé en binaire : 10111001. Ses 4 bits forts sont 1011.
De même pour l'*image11* : De 100001, nous ne retiendrons que 0010.
On va ensuite concaténer ces 8 bits, ce qui donnera 10110010.
De cette façon, en affichant la nouvelle image (*image2*), l'*image11* sera peu visible car cachée dans les bits de poids faibles.
Vous trouverez plus bas le programme *stegano2.py* qui permet de réaliser cette "insertion".
Dans les lignes 15 à 17, on fait un *ET* logique "(&)" avec le nombre 240, soit 11110000 en binaire. Ainsi, les 4 derniers bits sont mis à zéro. (Pour mettre à zéro les 2 derniers bits, on utiliserait 252).
Dans les lignes 20 à 22, on utilise "» n" qui permet de décaler de n bits vers la droite (pour décaler à gauche, on utiliserait "« n").
Dans les lignes 24 à 26, l'opérateur "|" (*OU* logique) permet d'insérer les pixels de l'*image11* dans l'*image10* pour obtenir finalement l'*image2*.

stegano2.py

```
#-*- coding: utf-8 -*-

from PIL import Image
image_billes = Image.open('billes.tif')
image_paysage = Image.open('paysage.tif')
largeur, hauteur = image_billes.size

image_finale = Image.new("RGB",(largeur, hauteur))

# Insertion de image_paysage dans image_billes
for ligne in range(hauteur):
    for colonne in range(largeur):
        pixel1 = image_billes.getpixel((colonne,ligne))
        rouge1 = pixel1[0] & 240 # opérateur "and"
        vert1 = pixel1[1] & 240  # on met à zéro les
        bleu1 = pixel1[2] & 240  # 4 derniers bits

        pixel2 = image_paysage.getpixel((colonne,ligne))
        rouge2 = pixel2[0] >> 4 # décalage de 4 bits
        vert2 = pixel2[1] >> 4  # vers la droite
        bleu2 = pixel2[2] >> 4

        rouge = rouge1 | rouge2 # opérateur "or"
        vert = vert1 | vert2    # qui permet de concaténer
        bleu = bleu1 | bleu2    # les 2 images

        image_finale.putpixel((colonne,ligne),(rouge,vert,bleu))
print(bin(pixel1[0])) # 0b10111001 bits forts de l'image10 : 1011
print(bin(pixel2[0])) # 0b100001   bits forts de l'image11: 0010
print(bin(rouge1))    # 0b10110000 image_billes
print(bin(rouge2))    # 0b10 image_paysage
print(bin(rouge))      # 0b10110010 bits de l'image_finale

image_finale.save('billes_incrust.tif')
image_billes.show()
image_paysage.show()
image_finale.show()
```

⇒ **Activité 13. 66**

Écrivez un programme, appelé par exemple *stegano3.py*, qui permet de réaliser l'opération inverse, c'est-à-dire de récupérer l'image cachée.

Le programme :

13.5.3 Traitement des images avec *numpy*

Dans cette partie, on importe le module *numpy* sous l'alias *np* et on récupère la matrice de l'objet image avec la fonction `np.array()`. Les *array* de *numpy* sont des tableaux multidimensionnels comme les listes de *Python* mais qui ne peuvent contenir que des objets du même type (*int*, *float*, *bool*, *complex*) contrairement aux listes. Enfin on aplatit cette matrice 2×2 avec la méthode `flatten()`.
Un exemple pour commencer avec le programme *image-numpy-1.py* :

image-numpy-1.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 24 21:10:23 2019

@author: marco5
"""

from PIL import Image
import numpy as np

mon_image = Image.open("billes.jpg")
data = mon_image.getdata()
print("data[0] : ", data[0])
print("len(data) : ",len(data))
image_np = np.array(mon_image)
largeur, hauteur, profondeur = image_np.shape
print("image_np.shape : ",image_np.shape)
print("largeur,hauteur : ",largeur,hauteur)
print(image_np[0][0],image_np[0][1],image_np[0][2])
```

```
21  data_plat = image_np.flatten()
22  print("data_plat : ",data_plat)
23  print("data_plat.shape : ",data_plat.shape)
24  print("data_plat[0] : ",data_plat[0])
25  mat1 = np.array([[12,12,15,28],[4,15,18,56],[14,5,11,26]])
26  print("mat1.shape : ",mat1.shape)
27  mat2 = np.zeros((3,6))
28  print("mat2 : ",mat2)
29  print("mat2.shape : ",mat2.shape)
30  img1=np.zeros((5,2,1),dtype=np.uint8)
31  img2=np.zeros((5,2,4),dtype=np.uint8)
32  print("img1 : ",img1)
33  print("img2 : ",img2)
```

Le résultat en console :

```
data[0] :  (176, 160, 135)
len(data) :  262144
image_np.shape :  (512, 512, 3)
largeur,hauteur :  512 512
[176 160 135] [175 159 134] [173 157 132]
data_plat :  [176 160 135 ... 146 127  95]
data_plat.shape :  (786432,)
data_plat[0] :  176
mat1.shape :  (3, 4)
mat2 :  [[0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0.]]
mat2.shape :  (3, 6)
img1 :  [[[0]
[0]]

[[0]
[0]]

[[0]
[0]]

[[0]
[0]]

[[0]
[0]]]
img2 :  [[[0 0 0 0]
[0 0 0 0]]

[[0 0 0 0]
[0 0 0 0]]

[[0 0 0 0]
[0 0 0 0]]

[[0 0 0 0]
[0 0 0 0]]

[[0 0 0 0]
[0 0 0 0]]]
```

On pourra également se reporter à la partie qui traite de ce module *numpy* dans le document distribué en première année.

On rappelle que la matrice d'une image couleur est semblable à la matrice d'une image en niveaux de gris, sauf que chaque pixel est représenté par un triplet de nombres entre 0 et 255 au lieu d'un nombre. Cette propriété est illustrée à la fin du programme précédent.
Par exemple, `image[i,j][k]` est un nombre entre 0 et 255 qui représente l'intensité du canal k ($k = 0$ pour le rouge, $k = 1$ pour le vert et $k = 2$ pour le bleu).
Pour créer une image couleur remplie de 0, il faut initialiser les triplets de couleur. On la déclarera avec :
`image = np.zeros([nb_lignes,nb_colonnes,3],dtype=np.uint8)`

Ainsi, pour convertir une image en niveaux de gris avec 3 canaux identiques (profondeur = 3), on pourra utiliser la fonction suivante :

```
def niveaux_de_gris (image):
  hauteur,largeur,profondeur = image.shape
  new_image=np.zeros((hauteur,largeur,profondeur),dtype=np.uint8)
  for i in range (hauteur) :
    for j in range (largeur) :
      new_image[i,j] = int(image[i,j][0]/3+image[i,j][1]/3+image[i,j][2]/3)
      # pb de dépassement de uint8 si division après la somme
  return new_image
```

On pourra visualiser l'image originelle avec :

```
import numpy as np
from PIL import Image

nom_image = "billes.jpg"
image = Image.open(nom_image)
mon_image = np.array(image)
image.show()
```

puis l'image obtenue avec :

```
ndg = niveaux_de_gris(mon_image)
nouvelle_image = Image.fromarray(ndg)
nouvelle_image.show()
```

Quelques opérations maintenant que vous devez réaliser en gardant pour chaque image les 3 canaux RGB et en travaillant bien entendu avec le module *numpy*...

13.5.3.1 Conversion en niveaux de gris avec la valeur du rouge

Écrire la fonction `image_rouge_gris` qui partant d'une image en couleurs renvoie une image en niveau de gris dont chaque pixel contient l'intensité en rouge de l'image précédente.
Par exemple si un pixel de l'image originelle a comme valeur $(203, 15, 23)$, la valeur du même pixel de l'image en niveaux de gris sera 203.

13.5.3.2 Conversion en rouge

Écrire la fonction `image_rouge` qui partant d'une image en couleurs renvoie une image en couleur dont chaque pixel ne contient que l'intensité en rouge de l'image originelle.
Par exemple si un pixel de l'image originelle vaut $(203, 15, 23)$, la valeur du même pixel de l'image en couleurs sera $(203, 0, 0)$.

13.5.3.3 Conversion en noir et blanc

Écrire la fonction `noir_et_blanc` qui partant d'une image en couleurs renvoie une image en noir (0) et blanc (255).
Par exemple si un pixel de l'image originelle vaut $(203, 15, 23)$, la valeur du même pixel de l'image en couleurs sera $(0, 0, 0)$. Si un pixel de l'image originelle vaut $(203, 180, 102)$, la valeur du même pixel de l'image en couleurs sera $(255, 255, 255)$.

13.5.3.4 Rotation trigonométrique

Écrire la fonction `rotation` qui effectue une rotation de $90°$ de l'image dans le sens trigonométrique.

13.5.3.5 Dilatation

Écrire la fonction dilatation qui prend en argument une image en noir et blanc et qui rend noir un pixel si au moins un de ses voisins est noir et le rend blanc sinon.

13.5.3.6 Érosion

Écrire la fonction `erosion` qui prend en argument une image en noir et blanc et qui rend noir un pixel si tous ses voisins sont noirs et le rend blanc sinon.

13.5.3.7 Détection de contours

Écrire la fonction `contour` qui prend en argument une image en niveaux de gris sur 3 canaux et détecte les contours par convolution. On prendra comme noyau la matrice suivante :

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Le programme :

13.6 Exercices

13.6.1 Palettes de couleurs

► Exercice 13. 29 : Palette de gris

Créez une palette en niveaux de gris, par exemple une bande de hauteur 100 pixels et de largeur 2560 pixels, allant du noir à gauche au blanc à droite.

Le résultat :

FIGURE 13.28 – palette_gris

► Exercice 13. 30 : Palette de couleurs

Créez une palette en couleurs, par exemple une bande de hauteur 256 pixels et de largeur 256 pixels. plusieurs possibilités existent : à vous de choisir.

Un résultat possible :

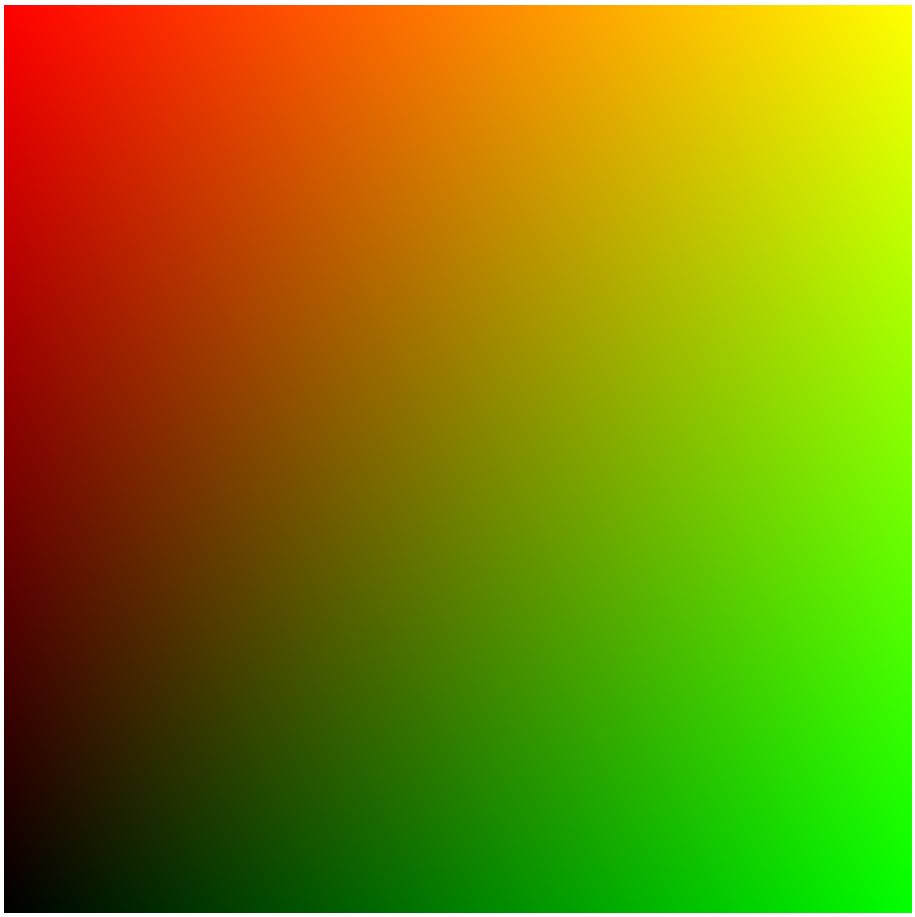


FIGURE 13.29 – palette-coul

13.6.2 Mosaïque de couleurs

► Exercice 13. 31 : Mosaïque de couleurs

Découpez une image en 9 rectangles et gardez un seul canal (ou 2) pour chaque rectangle.

Un résultat possible :

FIGURE 13.30 – mosaique

13.6.3 Fusion d'images

► Exercice 13. 32 : Fusion d'images

Ouvrez 2 images et si elles sont de tailles différentes, prenez la plus petite. Pour les fusionner, on peut prendre le maximum de chaque canal.

Un résultat possible :

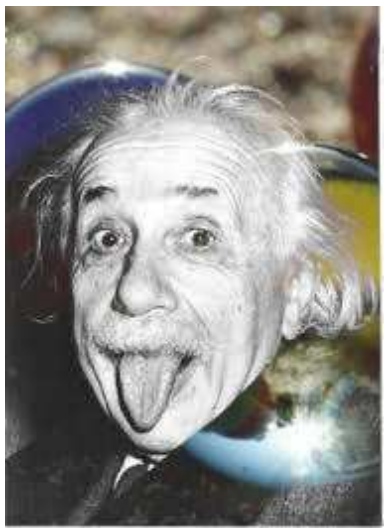


FIGURE 13.31 – fusion

13.6.4 Ajout d'une bordure

Principe

Pour créer une bordure, il faut créer une image plus grande, faire le cadre (par exemple blanc, donc avec des valeurs prises à 255), puis copier l'image initiale au milieu.

► Exercice 13. 33 : Cadre

Écrivez une fonction permettant de créer un cadre de largeur n pixels de la couleur de votre choix sur une image. Vous pourrez compléter le programme *bordure-eleve.py*.

bordure-eleve.py

```
1  # -*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  ext1='.jpg'
6  ext2='.eps'
7
8  def bordure(image,nb_bord=20,rouge=100,vert=150,bleu=250):
9
10     largeur,hauteur = image.size
11
12     new_largeur=.................
13     new_hauteur=.................
14
15     nouvelle_image = Image.new("RGB",\
16     (new_largeur,new_hauteur))
17
18     for colonne in range(new_largeur):
19         for ligne in range(new_hauteur):
20             nouvelle_image.putpixel((colonne,ligne),\
21             (rouge,vert,bleu))
22
23     for ligne in range(hauteur):
24         for colonne in range(largeur):
25             pix = image.getpixel((colonne,ligne))
26             nouvelle_image.putpixel(.................)
27
28     return(nouvelle_image)
29
30 im = "billes"
31 mon_image = Image.open(im+ext1)
32 ma_nouvelle_image = bordure(mon_image)
33 ma_nouvelle_image.save(im+'_bord'+ext2)
34 ma_nouvelle_image.show()
```

Un résultat possible :

FIGURE 13.32 – billes_bord

13.6.5 Pixelisation

Dans la suite, on traitera des images en niveaux de gris ou on les convertira, pour plus de facilité.

Principe

Pour pixeliser une image, on peut faire la moyenne par carrés de 4 pixels par exemple. Cela nécessite un nombre pair de pixels en largeur et en hauteur.

► Exercice 13. 34 : Pixelisation

Écrivez la fonction correspondante, en supposant que l'image de départ contient effectivement un nombre pair de pixels en largeur et en hauteur. Vous pourrez compléter le programme *pixelisation-eleve.py*.

pixelisation-eleve.py

```
1  #-*- coding: utf-8 -*-
2
3  from PIL import Image
4
5  # Si l'image contient un nombre pair de pixels
6  def pixelisation(image1):
7      image_gris = image1.convert('L')
8      largeur,hauteur = image_gris.size
9      pixel_gris = image_gris.load()
10     image2 = Image.new("L",(largeur,hauteur))
11     nouveau_pixel = image2.load()
12     for colonne in range (..............):
13         for ligne in range (..............):
14             moyenne = ............................
15             nouveau_pixel[colonne,ligne] = \
16             nouveau_pixel[colonne+1,ligne] = \
17             nouveau_pixel[colonne,ligne+1] = \
18             nouveau_pixel[colonne+1,ligne+1] = moyenne
19     return image2
20
21 nom_fichier = 'billes'
22 ext1 = '.jpg'
23 ext2 = '.eps'
24 mon_image=nom_fichier+ext1
25
26 image_originale = Image.open(mon_image)
27 image_pix = pixelisation(image_originale)
28 image_pix.save(nom_fichier+'_pixel'+ext1)
```

```
29 image_pix.save(nom_fichier+'_pixel'+ext2)
30 image_pix.show()
```

Le résultat :

FIGURE 13.33 – billes_pixel

13.6.6 Bruitage

Principe

Le bruit d'image est la présence d'informations parasites ajoutées de façon aléatoire à l'image.

► Exercice 13. 35 : Bruitage

Écrivez une fonction qui ajoute du bruit sur une image.

Vous pourrez utiliser le module *random* ainsi que *randint(a,b)* pour générer des entiers aléatoires compris entre a et b.

Pour ajouter du bruit, vous prendrez $[a,b] = [0,20]$ par exemple et vous créerez une **fonction**.

Le résultat :

FIGURE 13.34 – billes_bruit

13.6.7 Floutage

Principe

Le floutage peut se réaliser avec la méthode de filtrage vue page 188 avec la matrice :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

▶ Exercice 13. 36 : Floutage

Utilisez la méthode vue page 188 pour effectuer du floutage sur une image de votre choix. Vous créerez une **fonction**.

⚠ Il ne faudra pas oublier de ramener les valeurs dans le domaine visible puisque $s_c = 9 \ldots$

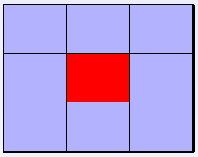
Ce que l'on obtient :

FIGURE 13.35 – billes_flou

13.6.8 Dilatation

Principe

La morphologie mathématique repose sur l'utilisation d'un élément structurant. Ce dernier est composé d'un pixel central (ici en rouge) et d'un ensemble de pixels (ici en bleu). Il se déplace sur la matrice *image_initiale* qui est en noir (bit de valeur 0) et blanc (bit de valeur 1) et lorsque le pixel central de l'élément structurant (en rouge) est sur un pixel *pix[x,y]* de l'image, un test est réalisé sur les pixels voisins (ici 8) pour déterminer la valeur du pixel *nouveau_pix* de la matrice *image_finale*.
On se donne ici un élément structurant simple :



L'ébauche de l'algorithme pourrait être comme suit :

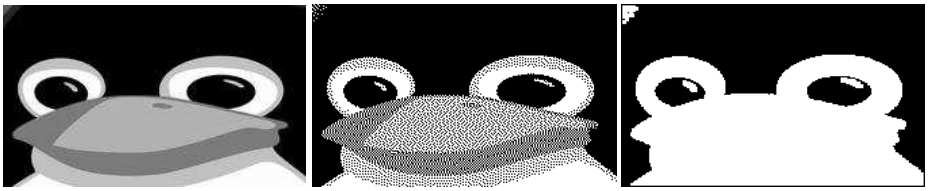
- Parcourir les pixels de l'*image_initiale*
- Pour chaque pixel *pix[x,y]* :
  - Centrer l'élément structurant sur ce pixel,
  - Considérer les 8 voisins du pixel *pix[x,y]*,
  - Si l'un de ces pixels est blanc, mettre *nouveau_pix* en blanc (bit de valeur 1).

Évidemment, les bords de l'image vont poser problème. L'astuce consiste à créer une bordure de largeur 1 pixel de couleur noire autour de l'*image_initiale*.

► Exercice 13. 37 : Dilatation

Proposez un programme permettant d'effectuer cette opération de dilatation, par exemple sur l'image *tux20_NetB.bmp*.

Les images obtenues :



(a) tux_20_gris (b) tux_20_NetB (c) tux_20_dilat

FIGURE 13.36 – dilatation

13.6.9 Érosion

Principe

On utilise la même technique que précédemment avec le même élément structurant.
L'ébauche de l'algorithme est très semblable :

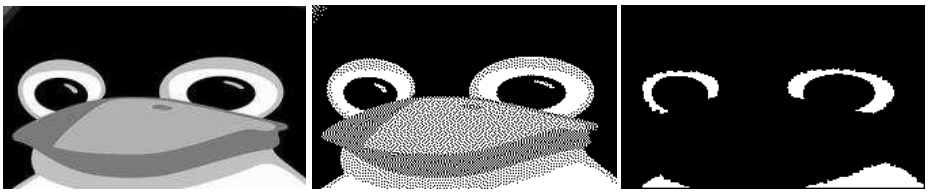
- Parcourir les pixels de l'*image_initiale*
- Pour chaque pixel *pix[x,y]* :
  - Centrer l'élément structurant sur ce pixel,
  - Considérer les 8 voisins du pixel *pix[x,y]*,
  - Si l'un de ces pixels est noir, mettre *nouveau_pix* en noir (bit de valeur 0).

Là encore, les bords de l'image vont poser problème. L'astuce consiste ici à créer une bordure de largeur 1 pixel de couleur blanche autour de l'*image_initiale*.

► Exercice 13. 38 : Érosion

Proposez un programme permettant d'effectuer cette opération d'érosion, par exemple sur l'image *tux20_NetB.bmp*.

Les images obtenues :



(a) tux_20_gris (b) tux_20_NetB (c) tux_20_erod

FIGURE 13.37 – érosion

13.6.10 Dilatation et érosion en niveaux de gris

Principe

La démarche est la même sauf que le pixel central de l'élément structurant prend la valeur maximale des pixels environnants pour la dilatation et la valeur minimale pour l'érosion.

► Exercice 13. 39 : Dilatation et érosion en gris

Modifiez les programmes précédents de façon à pouvoir dilater et éroder une image en niveaux de gris, par exemple l'image *tux20_gris.png*.

Les résultats :

(a) tux_20_gris (b) tux_20_dilat_bis (c) tux_20_erod_bis

FIGURE 13.38 – dilatation et érosion en niveaux de gris

13.7 Solutions des exercices

14

Cryptologie

14.1 Introduction

La cryptologie est la science des messages secrets.Elle se décompose en deux parties :

- la cryptographie, qui est l'art de crypter un message, c'est-à-dire de transformer un message intelligible en un autre incompréhensible pour le commun des mortels,
- la cryptanalyse, qui est l'art d'analyser un message crypté afin de le décrypter.

Nous allons étudier dans ce chapitre plusieurs façons de crypter un message et évidemment de le décrypter, voire de casser son cryptage.
Évidemment, il y a de multiples méthodes de cryptage, monoalphabétiques ou polyalphabétiques. Nous nous restreindrons ici à quelques méthodes monoalphabétiques.
Auparavant, nous formaterons le texte à crypter.
Nous allons également analyser les fréquences de lettres d'un texte.
Tous les programmes et fonctions seront ici écrites en *Python*.

Nous aurons notamment besoin dans cette étude des fonctions `ord()`, `chr()`, % et // :

```
>>> ord('A')
65
>>> ord('a')
97
>>> chr(66)
'B'
>>> for i in range(65,91):
...     print(chr(i),end=',')
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,
>>> for i in range(97,123):
...     print(chr(i),end=',')
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,
>>> 25%3
1
>>> 25//3
8
```

14.2 Formatage du texte

14.2.1 Gestion des accents, cédilles, ...

On peut effectuer le remplacement des lettres accentuées par exemple, dans une chaîne :

```
>>> ma_chaine = "arrêté grâce à Loïc
>>> ma_chaine.replace("ê","e").replace("ï","i")
'arreté grâce à Loic'
```

On peut également écrire une fonction pour remplacer toutes les lettres accentuées. On les place tout d'abord dans un dictionnaire :

```
dictionnaire = {"é" : 'e', "ê" : 'e', "è" : "e", "î" : "i", "ï" : "i", "ö" : "o", "ô"
: "o", "à" : "a", "â" : "a", "ù" : "u", "ç" : "c", "æ" : "ae", "œ" : "oe"}
```

Puis, pour chaque caractère accentué (ce sont les clés) de la chaîne `texte`, on le remplace par sa valeur :

crypt-01.py{11}{14}

```
1 def multi_replace(texte, dico):
2     for clef in dico:
3         texte = texte.replace(clef, dico[clef])
4     return texte
```

14.2.2 Élimination de la ponctuation et des espaces

Les lettres majuscules sont codées en ASCII de 65 à 90 inclus.

⇒ **Activité 14. 67**

Écrivez une fonction `epurage(chaine)` prenant un argument `chaine` qui permet de passer le message en majuscules (ou bien en minuscules si vous préférez) puis d'éliminer la ponctuation et des espaces dans un texte.

Le programme :

14.2.3 Séparation du texte en blocs

Souvent, on découpe les messages en cryptographie : les textes sont séparés en blocs de 4 ou 5 caractères.

⇒ **Activité 14. 68**

Écrivez une fonction `separe_bloc(chaine, bloc=4)` prenant en argument `chaine` ainsi qu'un argument optionnel `bloc` qui permet de découper le texte en blocs de n caractères (4 par défaut) en insérant un espace entre chaque bloc.

Le programme :

14.2.4 Opérations sur un fichier texte

Nous pouvons faire la même chose avec un fichier texte avec une extension `.txt`.
Pour le lire, on peut utiliser la fonction suivante :

```
def lecture(fichier):
  with open(fichier,'r') as source:
    resultat = source.read()
  return resultat # [:20] # si texte très long

message2 = lecture("poeme1.txt")
```

⇒ **Activité 14. 69**

Rassemblez ce qui précède dans un seul programme et le tester avec le fichier *poeme1.txt*.
Nous obtenons alors le programme *crypt-01.py* :

Le programme :

14.3 Outils pour le cryptologue

14.3.1 Analyse de fréquence

Principe

L'analyse de fréquence d'un texte permet, lorsque nous avons remplacé une lettre d'un texte par une autre, de reconnaître les lettres les plus fréquentes du texte, le "e" en français, par exemple.

Dans la langue française, en faisant abstraction des accents et suivant le type de langage utilisé, nous pouvons prendre les pourcentages suivants :

Lettre	A	B	C	D	E	F	G	H	I	J	K	L	M
Fréquence	8.40	1.06	3.03	4.18	17.26	1.12	1.27	0.92	7.34	0.31	0.05	6.01	2.96

Lettre	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Fréquence	7.13	5.26	3.01	0.99	6.55	8.08	7.07	5.74	1.32	0.04	0.45	0.30	0.12

Nous pourrons pour plus de précisions et d'autres langues, nous reporter au tableau 14.2 en fin de chapitre.

14.3.1.1 Fréquence d'une lettre de l'alphabet

Voici un algorithme qui permet de calculer la fréquence de la lettre `lettre` dans le texte `texte` :

Fréquence d'une lettre

```
VARIABLES
nbcar, cpt, i : entier
freq : réel
lettre : chaine
texte : chaine
DEBUT_ALGORITHME
   INITIALISATION
   nbcar ← longueur(texte)
   cpt ← 0
    POUR i ALLANT_DE 1 A nbcar
       DEBUT_POUR
       SI texte[i] = lettre ALORS
          DEBUT_SI
          cpt ← cpt + 1
          FIN_SI
       FIN_POUR
    freq ← cpt/nbcar
 FIN_ALGORITHME
```

14.3.1.2 Fréquence de l'ensemble des lettres de l'alphabet

Nous prendrons par exemple :

```
alphabet_min="abcdefghijklmnopqrstuvwxyz"
alphabet_maj="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

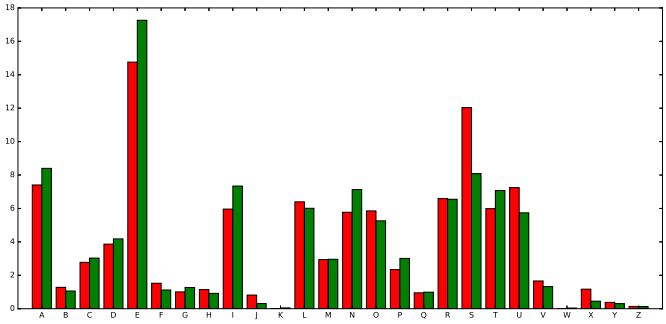
⇒ **Activité 14. 70**

Modifiez l'algorithme précédent de façon à calculer la fréquence de chacune des lettres et déduisez-en la fonction `freq_lettre(texte)` en *python* correspondante.

14.3.1.3 Illustration graphique

⇒ **Activité 14. 71**

Représentez, sous forme d'histogramme et sur un même graphe, les fréquences des lettres d'un texte et les fréquences de l'alphabet en général dans la langue française.
Nous obtenons par exemple :



Le programme :

14.3.2 Outils de cryptanalyse

14.3.2.1 Indice de coïncidence

Pour casser le code de Vigenère, nous aurons besoin de calculer ce que l'on appelle l'indice de coïncidence d'un texte. Son expression est la suivante :

$$I_C = \frac{\sum_{\lambda=A}^{\lambda=Z} f_\lambda\ (f_\lambda - 1)}{n\ (n-1)} = \frac{\sum_{\lambda=A}^{\lambda=Z} n_\lambda\ (n_\lambda - 1)}{n\ (n-1)}$$

où :

- n est le nombre total de lettres du message,
- n_λ (respectivement f_λ) est le nombre (respectivement la fréquence) de lettres λ dans le message.

En français, l'indice de coïncidence vaut environ $0{,}074$ (si en remplaçant les caractères accentués par les mêmes non accentués), en anglais, il vaut environ $0{,}065$ alors que pour un texte dont les lettres seraient distribuées de façon aléatoire, il faudrait $0{,}0385$.

Remarque

Le livre de Georges Perec "La disparition" ne contient pas la lettre "e"...

⇒ **Activité 14. 72**

En modifiant la fonction précédente `freq_lettre(texte)`, nous pouvons écrire une nouvelle fonction `indice_coinc(texte)` qui calcule l'indice de coïncidence d'un texte. C'est votre mission.

Le programme :

14.3.2.2 Indice de coïncidence mutuelle

Pour casser le code de *César*, nous aurons besoin de calculer ce que l'on appelle l'indice de coïncidence mutuelle de deux chaînes.
Soient deux chaînes `ch1` et `ch2` de longueurs respectives ℓ_1 et ℓ_2. L'indice de coïncidence mutuelle est la probabilité qu'un caractère aléatoire de `ch1` soit égal à un caractère aléatoire de `ch2`.
Son expression est alors la suivante :

$$I_{CM} = \frac{\sum_{i=0}^{25} f_{i_1} f_{i_2}}{\ell_1 \ell_2}$$

où :

- les f_{i_1} sont les fréquences des 26 caractères dans `ch1`,
- les f_{i_2} sont les fréquences des 26 caractères dans `ch2`.

On pourra prendre $\ell_2 = 1$.

⇒ **Activité 14. 73**

Écrivez une fonction `ind_coinc_mutuel(texte)` qui calcule cet indice en prenant pour une des chaînes l'alphabet avec comme fréquences :

```
freq_francais = [8.40, 1.06, 3.03, 4.18, 17.26, 1.12, 1.27, 0.92, 7.34, 0.31, 0.05,\
6.01, 2.96, 7.13, 5.26, 3.01, 0.99, 6.55, 8.08, 7.07, 5.74, 1.32, 0.04, 0.45, 0.30,
    0.12]
```

Vous pourrez utiliser la fonction `indice_coinc(texte)` précédemment définie. Vous pourrez également utiliser la fonction `zip` dont le comportement est illustré ci-dessous :

```
>>> list_1 = [1,2,3]
>>> list_2 = [3,4,5]
>>> sum(a*b for a,b in zip(list_1,list_2))
26
>>> sum(a+b for a,b in zip(list_1,list_2))
18
>>> sum(b/a for a,b in zip(list_1,list_2))
6.666666666666667
>>> sum(b-a for a,b in zip(list_1,list_2))
6
```

Le programme :

14.3.3 Outils mathématiques

Certains outils mathématiques sont indispensables en cryptologie [1].
Il est notamment souvent question de congruence *modulo n*, de *pgcd*, ...

14.3.3.1 Les nombres premiers

Nombres premiers entre eux

On dit que deux nombres entiers relatifs non nuls a et b sont premiers entre eux lorsqu'ils n'admettent pas de diviseur commun autre que le nombre 1.

1. Certaines parties mathématiques parmi les suivantes sont largement inspirées de la partie *Cryptographie* de Arnaud Bodin et François Recher.

Par conséquent :
On dit que deux nombres entiers relatifs non nuls a et b sont premiers entre eux lorsque leur pgcdest égal à 1.

Égalité de Bézout

Soit a et b deux entiers relatifs non nuls et D leur *PGDC*. Il existe deux entiers relatifs u et v tels que :

$$a\,u + b\,v = D$$

au ‘ bv “ D

Théorème de Bézout

Soit a et b deux entiers relatifs non nuls.

a et b sont premiers entre eux $\iff$ il existe deux entiers relatifs u et v tels que $a\,u + b\,v = 1$

14.3.3.2 Modulo

Soit $n \geqslant 2$ un entier fixé.

Modulo

On dit que a **est congru à** b **modulo** n, si n divise $b - a$. On note alors :

$$a \equiv b\ (n) \equiv b\ [n] \equiv b \pmod{n}$$

Pour nous $n = 26$. Ce qui fait que $28 \equiv 2 \pmod{26}$, car $28 - 2$ est bien divisible par 26. De même $85 = 3 \times 26 + 7$ donc $85 \equiv 7 \pmod{26}$.

On note $\mathbb{Z}/26\mathbb{Z}$ l’ensemble de tous les éléments de $\mathbb{Z}$ modulo 26. Cet ensemble peut par exemple être représenté par les 26 éléments $\{0, 1, 2, \ldots, 25\}$. En effet, puisqu’on compte modulo 26 :

$$0, 1, 2, \ldots, 25, \quad \text{puis} \quad 26 \equiv 0, 27 \equiv 1, 28 \equiv 2, \ldots, \quad 52 \equiv 0, 53 \equiv 1, \ldots$$

et de même $-1 \equiv 25, -2 \equiv 24, \ldots$

Plus généralement $\mathbb{Z}/n\mathbb{Z}$ contient n éléments. Pour un entier $a \in \mathbb{Z}$ quelconque, son **représentant** dans $\{0,1,2,\ldots,n-1\}$ s’obtient comme le reste k de la division euclidienne de a par n : $a = bn + k$. De sorte que $a \equiv k \pmod{n}$ et $0 \leq k < n$.

De façon naturelle l’addition et la multiplication d’entiers se transposent dans $\mathbb{Z}/n\mathbb{Z}$.

Pour $a,b \in \mathbb{Z}/n\mathbb{Z}$, on associe $a + b \in \mathbb{Z}/n\mathbb{Z}$.

Par exemple dans $\mathbb{Z}/26\mathbb{Z}$, $15 + 13$ égale 2. En effet $15 + 13 = 28 \equiv 2 \pmod{26}$. Autre exemple : que vaut $133 + 64$? $133 + 64 = 197 = 7 \times 26 + 15 \equiv 15 \pmod{26}$. Mais on pourrait procéder différemment : tout d’abord $133 = 5 \times 26 + 3 \equiv 3 \pmod{26}$ et $64 = 2 \times 26 + 12 \equiv 12 \pmod{26}$. Et maintenant sans calculs : $133 + 64 \equiv 3 + 12 \equiv 15 \pmod{26}$.

On fait de même pour la multiplication : pour $a,b \in \mathbb{Z}/n\mathbb{Z}$, on associe $a \times b \in \mathbb{Z}/n\mathbb{Z}$.

Par exemple 3×12 donne 10 modulo 26, car $3 \times 12 = 36 = 1 \times 26 + 10 \equiv 10 \pmod{26}$. De même : $3 \times 27 = 81 = 3 \times 26 + 3 \equiv 3 \pmod{26}$. Une autre façon de voir la même opération est d’écrire d’abord $27 = 1 \pmod{26}$ puis $3 \times 27 \equiv 3 \times 1 \equiv 3 \pmod{26}$.

14.3.3.3 Le petit théorème de Fermat amélioré

Voici le petit théorème de Fermat :

Petit théorème de Fermat

Si p est un nombre premier et $a \in \mathbb{Z}$ alors $a^p \equiv a \pmod p$

et sa variante :

Corollaire

Si p ne divise pas a alors : $a^{p-1} \equiv 1 \pmod p$

Nous allons voir une version améliorée de ce théorème dans le cas qui nous intéresse :

Petit théorème de Fermat amélioré

Soient p et q deux nombres premiers distincts et soit $n = pq$. Pour tout $a \in \mathbb{Z}$ tel que $\mathrm{pgcd(a, n)} = 1$ alors : $a^{(p-1)\,(q-1)} \equiv 1 \pmod n$

On note $\varphi\,(n) = (p-1)\,(q-1)$, la **fonction d'Euler**. L'hypothèse $\mathrm{pgcd(a, n)} = 1$ équivaut ici à ce que a ne soit divisible ni par p, ni par q. Par exemple pour $p = 5$, $q = 7$, $n = 35$ et $\varphi\,(n) = 4 \cdot 6 = 24$. Alors pour $a = 1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, \ldots$ on a bien $a^{24} \equiv 1 \pmod{35}$.

Démonstration

Notons $c = a^{(p-1)\,(q-1)}$. Calculons c modulo p :

$$c \equiv a^{(p-1)\,(q-1)} \equiv (a^{(p-1)})^{q-1} \equiv 1^{q-1} \equiv 1 \pmod p$$

où l'on appliquer le petit théorème de Fermat : $a^{p-1} \equiv 1 \pmod p$, car p ne divise pas a.
Calculons ce même c mais cette fois modulo q :

$$c \equiv a^{(p-1)\,(q-1)} \equiv (a^{(q-1)})^{p-1} \equiv 1^{p-1} \equiv 1 \pmod q$$

où l'on appliquer le petit théorème de Fermat : $a^{q-1} \equiv 1 \pmod q$, car q ne divise pas a.
Conclusion partielle : $c \equiv 1 \pmod p$ et $c \equiv 1 \pmod q$.

Nous allons en déduire que $c \equiv 1 \pmod{p\,q}$.
Comme $c \equiv 1 \pmod p$ alors il existe $\alpha \in \mathbb{Z}$ tel que $c = 1 + \alpha p$; comme $c \equiv 1 \pmod q$ alors il existe $\beta \in \mathbb{Z}$ tel que $c = 1 + \beta q$. Donc $c - 1 = \alpha p = \beta\, q$. De l'égalité $\alpha p = \beta\, q$, on tire que $p|\beta\, q$.
Comme p et q sont premiers entre eux (car ce sont des nombres premiers distincts) alors par le lemme de Gauss on en déduit que $p|\beta$. Il existe donc $\beta' \in \mathbb{Z}$ tel que $\beta = \beta'\, p$.
Ainsi $c = 1 + \beta\, q = 1 + \beta'\, p\, q$. Ce qui fait que $c \equiv 1 \pmod{p\,q}$, c'est exactement dire $a^{(p-1)\,(q-1)} \equiv 1 \pmod n$.

14.3.3.4 L'algorithme d'Euclide étendu

Nous avons déjà étudié l'algorithme d'Euclide qui repose sur le principe que $\mathrm{pgcd(a, b)} = \mathrm{pgcd(b, a \pmod b)}$.

Voici sa mise en œuvre informatique.

euclide.py{5}{8}

```python
def euclide(a,b):
    while b !=0 :
        a , b = b , a % b
    return a
```

On profite que *Python* assure les affectations simultanées, ce qui pour nous, correspond aux suites

$$\begin{cases} a_{i+1} & = & b_i \\ b_{i+1} & \equiv & a_i \pmod{b_i} \end{cases}$$

initialisée par $a_0 = a$, $b_0 = b$.

Nous avons vu aussi comment « remonter » l'algorithme d'Euclide à la main pour obtenir les cœfficients de Bézout u, v tels que $a\,u + b\,v = \text{pgcd(a, b)}$. Cependant il nous faut une méthode plus automatique pour obtenir ces cœfficients, c'est l'**algorithme d'Euclide étendu**.

On définit deux suites (x_i), (y_i) qui vont aboutir aux cœfficients de Bézout.

L'initialisation est :

$$x_0 = 1 \qquad x_1 = 0 \qquad y_0 = 0 \qquad y_1 = 1$$

et la formule de récurrence pour $i \geq 1$:

$$x_{i+1} = x_{i-1} - q_i x_i \qquad y_{i+1} = y_{i-1} - q_i y_i$$

où q_i est le quotient de la division euclidienne de a_i par b_i.

euclide.py{14}{22}

```
6  def euclide_etendu(a,b):
7      x = 1 ; xx = 0
8      y = 0 ; yy = 1
9      while b !=0 :
10         q = a // b
11         a , b = b , a % b
12         xx , x = x - q*xx , xx
13         yy , y = y - q*yy , yy
14     return (a,x,y)
```

Cet algorithme renvoie d'abord le *pgcd*, puis les cœfficients u,v tels que $a\,u + b\,v = \text{pgcd(a, b)}$.

14.3.3.5 Inverse modulo n

Soit $a \in \mathbb{Z}$, on dit que $x \in \mathbb{Z}$ est un **inverse de** a **modulo** n si $a\,x \equiv 1 \pmod{n}$.

Trouver un inverse de a modulo n est donc un cas particulier de l'équation $ax \equiv b \pmod{n}$.

- a admet un inverse modulo n si et seulement si a et n sont premiers entre eux.
- Si $a\,u + n\,v = 1$ alors u est un inverse de a modulo n.

En d'autres termes, trouver un inverse de a modulo n revient à calculer les cœfficients de Bézout associés à la paire (a, n).

Démonstration

La preuve est essentiellement une reformulation du théorème de Bézout :

$$\begin{aligned} \text{pgcd(a, n)} = 1 &\iff \exists\, u, v \in \mathbb{Z} \qquad a\,u + n\,v = 1 \\ &\iff \exists\, u \in \mathbb{Z} \qquad au \equiv 1 \pmod{n} \end{aligned}$$

Voici le code :

euclide.py{32}{37}

```
16 def inverse(a,n):
17     c,u,v = euclide_etendu(a,n)
```

```
18      if c != 1 :           # Si pgcd différent de 1 renvoie 0
19          return 0
20      else :
21          return u % n      # Renvoie l'inverse
```


14.3.3.6 Exponentiation rapide

Cette partie a déjà été traitée en cours dans le chapitre sur la complexité.

1. Méthode naïve

   (a) Algorithmique :

   Calculer a^n nécessite a priori $n-1$ multiplications selon l'algorithme suivant :

   Exponentiation naïve

   ```
   VARIABLES
   a : float
   n : int
   ENTRÉES
   un réel a et une puissance n
   INITIALISATION
   Resultat ← a
   DEBUT_ALGORITHME
      POUR k ALLANT_DE 2 A n
          DEBUT_POUR
          Resultat ← Resultat × a
          FIN_POUR
       AFFICHER Resultat
    FIN_ALGORITHME
   ```

   (b) Avec *Python* :

   puiss-naive.py{3}{7}

   ```
   def puissance_naive(a,n):
       res=a
       for k in range(1,n):
           res*=a
       return res
   ```

2. Algorithme de HÖRNER

   (a) Conventions :

   Dans la suite, nous confondrons polynôme et fonction polynôme.

   (b) Principe :

   Prenons l'exemple de $P(x)=3x^5-2x^4+7x^3+2x^2+5x-3$. Le calcul classique nécessite 5 additions et 15 multiplications.

   On peut faire pas mal d'économies de calcul en suivant le schéma suivant :

$$\begin{aligned}
P(x) &= \underbrace{a_n x^n + \cdots + a_2 x^2 + a_1 x}_{\text{on met } x \text{ en facteur}} + a_0 \\
&= \left(\underbrace{a_n x^{n-1} + \cdots + a_2 x}_{\text{on met } x \text{ en facteur}} + a_1 \right) x + a_0 \\
&= \cdots \\
&= (\cdots(((a_n x + a_{n-1}) x + a_{n-2}) x + a_{n-3}) x + \cdots) x + a_0
\end{aligned}$$

   Ici cela donne $P(x)=(((((3x)-2)x+7)x+2)x+5)x-3$ c'est-à-dire 5 multiplications et 5 additions. En fait il y a au maximum 2 fois le degré de P opérations (voire moins avec les zéros).

(c) L'algorithme :

On peut essayer de faire mieux.

Voyons une première méthode qui utilise l'algorithme de HÖRNER étudié précédemment et la décomposition en base 2 de l'exposant.

Par exemple, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$. On peut donc associer à cette écriture un polynôme P tel que 11 soit égal à $P(2)$:

$$P(X) = 1 \times X^3 + 0 \times X^2 + 1 \times X^1 + 1 \times X^0 = ((X+0)X+1)X+1$$

Donc

$$\begin{aligned} a^{P(2)} &= a^{1+2(1+2(2\times 1+0))} \\ &= a \times a^{2(1+2(2\times 1+0))} \\ &= a \times \left(a^{1+2(2\times 1+0)}\right)^2 = a^1 \times \left(a^1 \times \left(a^0 \times \left(a^1\right)^2\right)^2\right)^2 \\ &= \left(\left(\left(a^1\right)^2 a^0\right)^2 a^1\right)^2 a^1 \end{aligned}$$

On en déduit l'algorithme :

Exponentiation « à la HÖRNER »

```
1   VARIABLES
2   a : float
3   n : int
4   ENTRÉES
5   un réel a et un entier n
6   INITIALISATION
7   C ← décomposition de n en base 2
8   res ← a
9   DEBUT_ALGORITHME
10      POUR j ALLANT_DE 1 A taille de C − 1
11         DEBUT_POUR
12         res ← res × res
13         SI le j-ème chiffre de la décomposition en base 2 est 1 ALORS
14            DEBUT_SI
15            res ← a × res
16            FIN_SI
17         AFFICHER res
18         FIN_POUR
19   FIN_ALGORITHME
```

Pour calculer a^{11} nous n'avons donc plus à effectuer que 6 multiplications ou plutôt 5 si on ne compte pas la multiplication par 1.

(d) Avec Python :

puiss-horner.py{3}{19}

```
1  def base2(n):
2      r=n%2
3      q=n//2
4      R=[r]
5      while q>0 :
6          r=q%2
7          q=q//2
8          R=[r]+R
9      return R
10
11 def puissance_horner(a,n):
12     C=base2(n)
13     res=a
14     for j in range(1,len(C)):
15         res*=res
16         if C[j]==1: res*=a
```

```
17        return(res)
```

3. Variante sans utiliser l'écriture en base 2

(a) Algorithme :

Voyons comment procéder sur notre exemple habituel :

$a^{11} = a \times a^{10} = a \times \left(a^5\right)^2 = a \times \left(a \times a^4\right)^2 = a \times \left(a \times \left(a^2\right)^2\right)^2$

Ainsi, on réduit l'exposant k selon sa parité :

- si k est pair, on écrit $a^k = \left(a^{\frac{k}{2}}\right)^2$;
- sinon, $a^k = a \times \left(a^{\frac{k-1}{2}}\right)^2$

Exponentiation rapide sans utiliser la base 2

```
1   VARIABLES
2   a : float
3   n : int
4   ENTRÉES
5   un réel a et un entier n
6   INITIALISATION
7   res ← 1
8   puissance ← n
9   temp ← a
10   DEBUT_ALGORITHME
11      TANT_QUE puissance non nulle FAIRE
12         DEBUT_TANT_QUE
13         SI puissance est impaire ALORS
14            DEBUT_SI
15            res ← temp × res
16            puissance ← puissance − 1
17            FIN_SI
18         puissance ← puissance/2
19         temp ← temp × temp
20         FIN_TANT_QUE
21   FIN_ALGORITHME
```

(b) Avec *Python* :

puiss-rapid.py{3}{13}

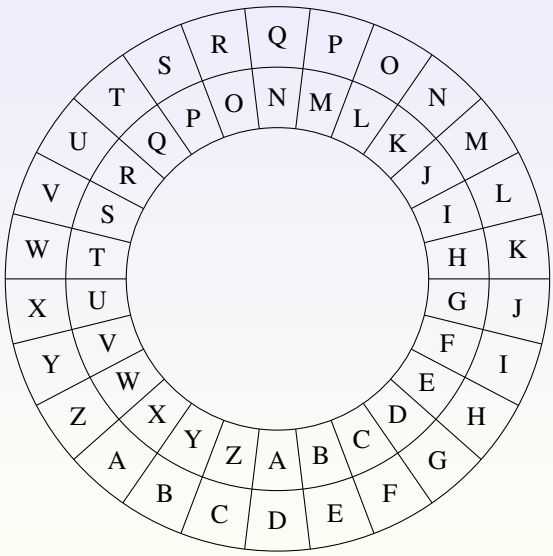
```
1  def expo_rapid(a,n):
2      res=1
3      Puissance=n
4      temp=a
5      while Puissance>0:
6          if Puissance%2==1 :
7              res*=temp
8              Puissance+=-1
9          Puissance=Puissance/2
10         temp*=temp
11     return(res)
```

14.4 Chiffrement de *César*

Jules César était un général, homme politique et écrivain romain, né à Rome le 12 (ou 13) juillet 100 avant *J.-C.* et mort le 15 mars 44 avant *J.-C.*.

Principe

Jules César, pour ses communications importantes, cryptait ses messages. Le chiffrement de *César* est un simple décalage de lettres. Par exemple, avec un décalage de 3, A devient D, B est remplacé par E, C par F, ...
On peut utiliser , pour crypter et décrypter, deux alphabets sur deux anneaux concentriques, qui peuvent tourner l'un par rapport à l'autre :



14.4.1 Cryptage

⇒ **Activité 14.74**

Écrivez une fonction `cesar_chiffre_lettre(lettre, dec)` qui code une lettre `lettre` avec le décalage `dec`.
Déduisez-en ensuite une fonction `cesar_chiffre_texte(texte, k)` (qui fait appel à la première fonction), qui décale tout un texte.

Le programme :

14.4.2 Décryptage

⇒ **Activité 14. 75**

Écrivez une fonction `cesar_dechiffre_lettre(x, dec)` qui code une lettre `lettre` avec le décalage `dec`.

Le programme :

14.4.3 Attaque du chiffrement de *César*

14.4.3.1 Attaque par force brute

Nous pouvons effectuer une attaque par force brute du code de *César* car il n'y a que 26 possibilités de chiffrement.

⇒ **Activité 14. 76**

Écrivez une procédure `cesar_dechiffre_texte(texte)` qui déchiffre un texte avec toutes les combinaisons possibles.

Le programme :

14.4.3.2 Attaque par analyse fréquentielle

Nous pouvons également, si le texte est suffisamment long, effectuer une analyse fréquentielle, puis, en calculant l'indice de coïncidence mutuelle, en déduire la clé (valeur du décalage de *César*).

⇒ **Activité 14. 77**

Écrivez une fonction `recherche_clef(texte)` qui retourne la valeur du décalage de *César* d'un texte crypté par calcul de l'indice de coïncidence mutuel. En effet, il est maximal pour le bon décalage. Il suffit donc de le calculer pour chaque décalage et de retenir la bonne valeur.

Le programme :

Vous pouvez alors décrypter le message codé `message2_code` en utilisant :

```
cle = recherche_clef(message2_code)
cesar_chiffre_texte(message2_code,-cle)}
```

14.5 Chiffrement affine

Comme le code de *César* est très simple, il est relativement facile à décrypter. C'est la raison pour laquelle on complique un peu les choses avec le codage dit affine. Le principe est le suivant :

Codage affine

- On remplace les lettres par des nombres allant de 0 (pour le A) à 25 (pour le Z).
- On choisit f une fonction affine, donc telle que $f(x) = a\,x + b$, avec a et b entiers compris entre 1 et 25.

Chaque lettre A, B, ... est codée par son rang entre 0 et 25.
On choisit un couple de nombres a et b (on peut se limiter à l'intervalle $[0, 25]$ car on retrouve ensuite les mêmes résultats.
En notant x le rang d'une lettre $x \in [0, 25]$ et $r(x)$ le reste de la division euclidienne de $y = a\,x + b$ par 26, $r(x)$ est la lettre correspondante codée.

Exemple 1

Choisissons par exemple f telle que $f(x) = 3\,x + 2$.

- Pour crypter la lettre C (donc le nombre 2), par exemple, on calcule $f(2) = 3 \times 2 + 2 = 8$, ce qui correspond à la lettre I.
- Pour crypter la lettre U (donc le nombre 20), par exemple, on calcule $f(20) = 3 \times 20 + 2 = 62$. Comme le résultat est supérieur à 25, on doit faire la division euclidienne de 62 par 26, ce qui donne $62 = 2 \times 26 + 10$ et la lettre correspond au reste, ici 10. On trouve alors la lettre K.

Exemple 2

$a = 17$, $b = 2$.
La lettre A, de rang 0 est codée par le nombre $0 \times 17 + 2 = 2$, qui correspond à la lettre codée C. La lettre K, de rang 10 est codée par le nombre $10 \times 17 + 2 = 172 = 16 + 6 \times 26 \equiv 16 \pmod{26}$, qui correspond à la lettre codée Q.

⇒ **Activité 14. 78**

1. Expliquez pourquoi le couple $(2, 0)$ ne convient pas pour effectuer un chiffrement affine.
2. Le couple $(3, 0)$ convient-il ?

Ainsi, si a n'est pas premier avec 26, deux lettres différentes de rang x et x' peuvent être chiffrées par la même lettre.
Dans ce cas, le déchiffrement est impossible.

Par contre, si a est premier avec 26, donc si $pgcd(a, 26) = 1$, deux lettres différentes de rang x et x' ne peuvent posséder le même codage. Le chiffrement est une bijection et le déchiffrement est possible.

En conséquence, **a doit être premier avec 26**.
Il y a ainsi pour le chiffrement affine 311 clés utilisables.

⇒ **Activité 14. 79**

Retrouver le nombre de clés utilisables pour le chiffrement affine.

14.5.1 Fonctions préliminaires

⇒ **Activité 14. 80**

Écrivez une fonction `code(texte)` qui, à partir d'une chaîne `texte`, retourne une liste dans laquelle chaque lettre du texte est remplacée par son numéro dans l'alphabet.

Le programme :

⇒ **Activité 14.81**

Écrivez une fonction `encode(liste)` qui, à partir d'une liste `liste`, retourne une chaîne dans laquelle chaque numéro est remplacé par la lettre majuscule correspondante dans l'alphabet.

Le programme :

14.5.2 Cryptage

Nous allons maintenant crypter un message avec une fonction du type $y = f(x) = a\,x + b$.

⇒ **Activité 14.82**

Écrivez une fonction `code_affine(texte, couple)` qui code un texte par fonction affine avec le couple `couple = (a, b)`. Nous pourrons accéder aux éléments a et b du tuple *couple* par :

```
a, b = couple[0], couple[1]
```

Le programme :

14.5.3 Décryptage

Pour déchiffrer un message, il faut procéder de la même façon. On commence par transcrire le message en nombres. Pour chaque nombre, on doit inverser la relation $y = a\,x + b$ (ici, on connaît y et on doit retrouver x). On a envie de poser $x = \dfrac{y}{a} - \dfrac{b}{a}$. C'est presque cela, sauf que l'on fait de l'arithmétique modulo 26. Ce qui remplace $\dfrac{1}{a}$, c'est l'inverse de a modulo 26, autrement dit un entier a' tel que, lorsqu'on fait le produit $a\,a'$, on trouve un entier de la forme $1 + 26\,k$. On sait qu'un tel entier existe si a et 26 sont premiers entre eux. Par exemple, pour $a = 3$, on peut choisir $a' = 9$ car $9 \times 3 = 1 + 26$.

Cette valeur de a' déterminée, on a alors $x = a'\,y - a'\,b$, qu'on retranscrit en une lettre comme pour l'algorithme de chiffrement.

Déchiffrement affine

Pour décoder le message, on peut appliquer le même principe avec un chiffrement affine dont la clé est le couple (a', b') tels que :

$$\begin{cases} a\,a' \equiv 1\ [26]\ :\ a' \text{ est l'inverse de } a \text{ modulo } 26 \\ b' \text{ est le reste de la division euclidienne de } a'\ (26 - b)\ \text{ par } 26 \end{cases}$$

Exemple 1

Reprenons le couple $(a, b) = (17, 2)$.
L'inverse de $a = 17$ modulo 26 est $a' = 23$.
De plus :

$$a'\,(26 - b) \equiv 6 \pmod{26}$$

qui donne $b' = 6$.
Ainsi, la lettre, codée S, correspondant au rang 18 correspond à la lettre originelle donnée par :

$$23 \times 18 + 6 \equiv 4 \pmod{26}$$

C'est la lettre de rang 4, donc E.

Exemple 2

Un message a été codé par un chiffrement affine. On note f la fonction de codage : $f(x)$ est le reste de la division euclidienne de $a\,x + b$ par 26. À l'aide d'une analyse fréquentielle, on a remarqué que la lettre E est codée par E et que la lettre T est codée par Z.

- E est codé par E donc $a \times 4 + b \equiv 4\ [26]$.
  T est codé par Z donc $a \times 19 + b \equiv 25\ [26]$.
  a et b vérifient donc le système :

$$\begin{cases} 4\,a + b \equiv 4\ [26] & (1) \\ 19\,a + b \equiv 25\ [26] & (2) \end{cases}$$

- En effectuant la soustraction (2) − (1) membre à membre, on en déduit :

$$15\,a \equiv 21\ [26]$$

- Pour déterminer l'inverse de 15 modulo 26, on écrit la relation de Bézout :

$$15 \times 7 - 26 \times 4 = 1$$

  Donc $15 \times 7 \equiv 1\ [26]$.
  De $15\,a \equiv 21\ [26]$, on tire :
  $7 \times 15\,a \equiv 7 \times 21\ [26]$, soit :
  $a \equiv 17\ [26]$.
  Donc $a = 17$ puisque $0 \leqslant a \leqslant 25$.
  On obtient alors :

$$\begin{cases} 68 + b \equiv 4\ [26] \\ 323 + b \equiv 25\ [26] \end{cases} \Longrightarrow b \equiv 14\ [26]$$

  Finalement, $(a, b) = (17, 14)$.
- Pour déterminer la fonction de décodage, on cherche tout d'abord l'inverse de 17 modulo 26.
  On trouve $17 \times (-3) + 26 \times 2 = 1$.
  On a $f(x) \equiv 17\,x + 14\ [26]$, donc :
  $-3\,f(x) \equiv x - 42\ [26]$, soit :
  $x \equiv -3\,f(x) + 42\ [26] \equiv 23\,f(x) + 16$.
  Pour décoder, il suffit de coder le message crypté à l'aide du codage affine $23\,x + 16$.

Exemple 3

On considère la phrase *LACLEESTSOUSLEPAILLASSON* que nous codons à l'aide du codage affine $y = 7\,x + 10 \pmod{26}$.
Nous obtenons alors *JKYJMMGNGEUGJMLKOJJKGGEX*.

- Pour trouver a' (inverse de a modulo 26), une des deux clés du décodage, on va utiliser l'algorithme de Bézout :

  $$7\,a' \equiv 1 \pmod{26}$$
  $$7\,a' = 1 + 26\,k$$

  Soit :

  $$-26\,k + 7\,a' = 1$$

  En posant $u = -k$ et $v = a'$, on obtient :

  $$26\,u + 7\,v = 1$$

- En utilisant l'algorithme de Bézout, on obtient :

  $$v = a' = -11 \equiv 15 \pmod{26}$$

- On obtient b' grâce à la relation :

  $$b' = -a'\,b = 11 \times 10 = 110 \equiv 6 \pmod{26}$$

- Ainsi, la transformation affine de décodage est :

  $$y = 15\,x + 6 \pmod{26}$$

⇒ **Activité 14. 83**

Le déchiffrement affine peut alors s'effectuer en créant une fonction `decode_affine(texte,couple)`. Celle-ci fait appel à la fonction `inverse(a,n)` définie dans les outils mathématiques ainsi qu'aux fonctions `code(texte)` et `decode(liste)`. À vous de jouer !

Le programme :

14.5.4 Casser un chiffrement affine

Supposons que nous ayons le message suivant codé au moyen d'une transformation affine. :
DIADSSOXAIATSSOEGODSNICDDIOOEVEGOEGODIBSVSXFS.
En français, la lettre la plus fréquente est le E suivi du S puis du A.
Avec un peu de chance, cet ordre fréquentiel va être suivi, à quelque chose près, par les lettres du texte à décoder : ceci sera d'autant plus vrai que le texte sera long et, également d'autant plus que le texte à décoder appartiendra à une famille analogue aux textes ayant servi à établir la table des fréquences. Ici, dans le court message que nous avons à décrypter, la lettre la plus fréquente est le S (18,75 %), suivi du O (14,6 %) :

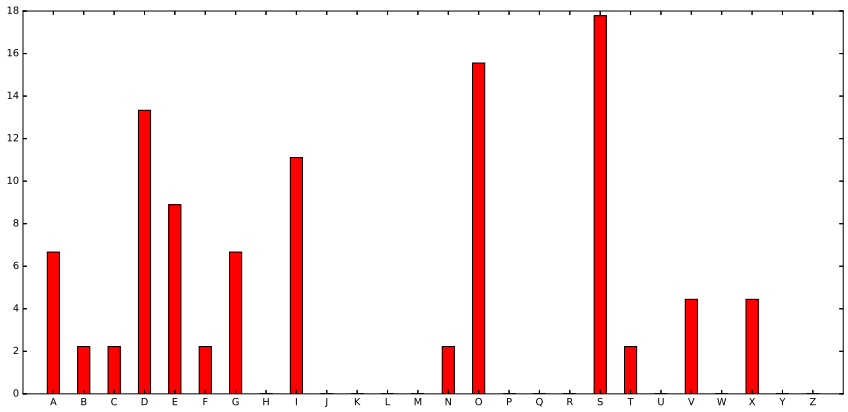


FIGURE 14.1 – affine1.eps

Faisons alors l'hypothèse que le S correspond au E et le O au S :
Passons aux équivalents numériques :

$$\begin{cases} S & \to 18 & \rightsquigarrow 4 & \to & \text{E} \\ O & \to 14 & \rightsquigarrow 18 & \to & \text{S} \end{cases}$$

Les paramètres de décryptage a' et b' doivent alors vérifier le système de deux équations :

$$\begin{cases} 18\,a' + b' & \equiv 4\ [26] & S \rightsquigarrow E & (3) \\ 14\,a' + b' & \equiv 18\ [26] & O \rightsquigarrow S & (4) \end{cases} \Longrightarrow 4\,a' \equiv -14\ [26] \equiv 12\ [26]$$

Les règles élémentaires de calcul sur des congruences de même module nous permettent d'opérer quasiment comme pour la résolution de systèmes d'équations classiques : nous ne pouvons faire de divisions, mais nous pouvons faire des multiplications.
On en déduit, avec (3) − (4) :

$$4\,a' \equiv -14 \pmod{26} \equiv 12 \pmod{26}$$

ou encore :

$$2\,a' \equiv -7 \pmod{13} \equiv 6 \pmod{13}$$

- Si a' vérifie cette équation, il existe $k \in \mathbb{Z}$ tel que :

$$2\,a' = -7 + 13\,k$$

- Multiplions par 7 qui est l'inverse de 2 modulo 13 :

$$\begin{aligned} 2\,a' \times 7 &= -7 \times 7 + 13 \times 7\,k \\ 14\,a' &\equiv -49 \pmod{13} \\ a' &\equiv -10 \pmod{13} \\ a' &\equiv 3 \pmod{13} \end{aligned}$$

Donc $a' \in \{3, 16, 29, \ldots\}$.
Modulo 26, cela nous donne $a' \equiv 3 \pmod{26}$ ou $a' \equiv 16 \pmod{26}$ (29 n'est pas dans l'intervalle).
a' doit être premier avec 26 pour être admissible, les deux options restent possibles.
De (3) ou (4), on déduit :
$b' = 4 - 18 \times 3 \equiv -50 \pmod{26} \equiv 2 \pmod{26}$ ou bien $b' = 18 - 16 \times 14 \equiv -206 \pmod{26} \equiv 2 \pmod{26}$.
Nous retrouvons ainsi la transformation affine de décodage avec :

```
print(code_affine(ch3_code,(3,2)))
```

et nous obtenons :
LACLEESTCACHEESOUSLEPAILLASSONOUSOUSLAFENETRE.
Nous vérifions l'hypothèse que le S correspond au E et le O au S grâce à l'histogramme du message originel :

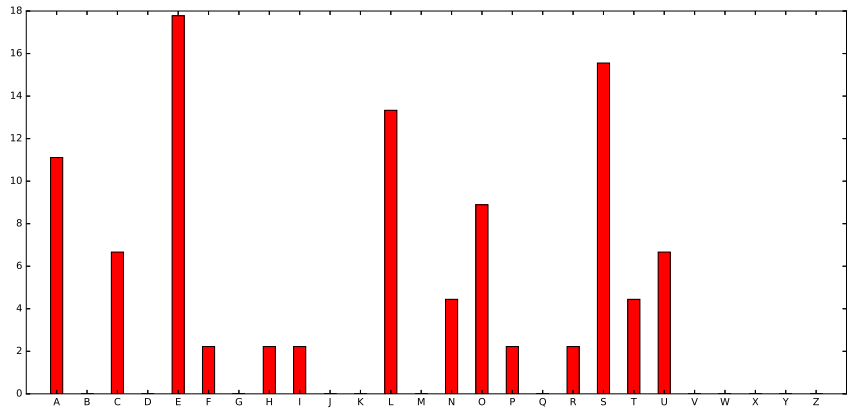


FIGURE 14.2 – affine2.eps

Cet exemple montre qu'une analyse statistique des fréquences de lettres permet facilement de briser un cryptogramme quand on sait que celui-ci est le fruit d'une transformation affine monoalphabétique.

Nous pourrions effectuer ces tâches avec *Python* mais une méthode de chiffrement plus intéressante nous attend.

14.6 Chiffrement de Vigenère

Même si l'on connaissait depuis fort longtemps les faiblesses de la cryptographie par substitution, et malgré les essais notamment d'*Alberti*, de *Porta* et de *Trithème*, il n'y eut pas entre *César* et le $XVI^{\text{è}}$ siècle de véritable nouveau procédé cryptographique, à la fois sûr (pour les moyens de l'époque) et facile à utiliser ! *Blaise de Vigenère* (1523-1596), fut l'initiateur d'une nouvelle façon de chiffrer les messages qui mit en échec les cryptanalystes trois siècles durant. *Vigenère* était un personnage à multiples facettes, tantôt alchimiste, écrivain, historien, il était aussi diplomate au service des ducs de Nevers et des rois de France. C'est en 1549 que *Vigenère* fut envoyé à Rome pour une affaire d'Etat. Durant son voyage, il se passionna pour les écrits de *Léone Batista Alberti*, *Jean Trithème* et *Giovanni Battista della Porta*, des cryptographes contemporains à son époque. Ses intérêts pour la cryptographie à cette époque étaient purement professionnels. Vers 1560, *Vigenère* se jugeant à l'abri du besoin, se consacra uniquement à la reprise des travaux d'*Alberti*, de *Trithème* et de *Porta*. C'est en 1586 qu'il publie son Traité des chiffres ou Secrètes manières d'écrire, qui explique son nouveau chiffre.
L'idée de *Vigenère* est d'utiliser un chiffre de *César*, mais où le décalage utilisé change de lettres en lettres.

Principe

Choisissons un mot "clef" qui va servir à coder le message, par exemple la clef *PREPATSI* pour coder le message *LA CRYPTOGRAPHIE NOUS EMBALLE*.

Lettre originelle	L	A	C	R	Y	P	T	O	G	R	A	P	H
Nombre originel	11	0	2	17	24	15	19	14	6	17	0	15	7
Clef	P	R	E	P	A	T	S	I	P	R	E	P	A
Décalage	15	17	4	15	0	19	18	8	15	17	4	15	0
Nombre codé	0	17	6	6	24	8	11	22	21	8	4	4	7
Lettre codée	A	R	G	G	Y	I	L	W	V	I	E	E	H

Lettre originelle	I	E	N	O	U	S	E	M	B	A	L	L	E
Nombre originel	8	4	13	14	20	18	4	12	1	0	11	11	4
Clef	T	S	I	P	R	E	P	A	T	S	I	P	R
Décalage	19	18	8	15	17	4	15	0	19	18	8	15	17
Nombre codé	1	22	21	3	11	22	19	12	20	18	19	0	21
Lettre codée	B	W	V	D	L	W	T	M	U	S	T	A	V

La clef est répétée autant de fois qu'il le faut en dessous du message.
Appliquons-lui ensuite un décalage "de *César*" en fonction de la lettre de la clef.
La première lettre L doit être codée avec P (p dans le carré de Vigenère). Appliquons-lui un décalage de 15, ce qui donne A.
La deuxième lettre A doit être codée avec R (r dans le carré de Vigenère). Appliquons-lui un décalage de 17, ce qui donne R.
La troisième lettre C doit être codée avec E (e dans le carré de Vigenère). Appliquons-lui un décalage de 4, ce qui donne G.
Le message codé est donc : *ARGG YILW VIEE HBWV DLWT MUST AV*.
On constate qu'une même lettre peut être codée de plusieurs façons, ce qui rend inefficace l'analyse des fréquences pour décrypter!
Pour coder-décoder, nous pouvons utiliser un carré de Vigenère (donné ci-dessous). Les trois premiers caractères y sont mis en valeur.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G[3]	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
ℓ	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A[1]	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R[2]	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

TABLE 14.1 – Carré de Vigenère

14.6.1 Cryptage

Le cryptage par la méthode de Vigenère est très simple.

⇒ **Activité 14. 84**
Écrivez une fonction `codage_vig1(message, clef)` qui chiffre une chaîne `message` à l'aide de la clé `clef` par la méthode de Vigenère.

Le programme :

1. Première lettre du message
2. Deuxième lettre
3. Troisième lettre

14.6.2 Décryptage

14.6.2.1 Decryptage avec la clé

Le décryptage n'est pas non plus très compliqué si la clé est connue.

⇒ **Activité 14. 85**

Écrivez une fonction `decodage_vig1(message_code, clef)` qui déchiffre une chaîne `message_code` à l'aide de la clé `clef` par la méthode de Vigenère.

Le programme :

14.6.2.2 Détermination de la clé

Nous voyons bien que l'histogramme n'a plus rien à voir avec celui d'une substitution simple : il est beaucoup plus "plat" :

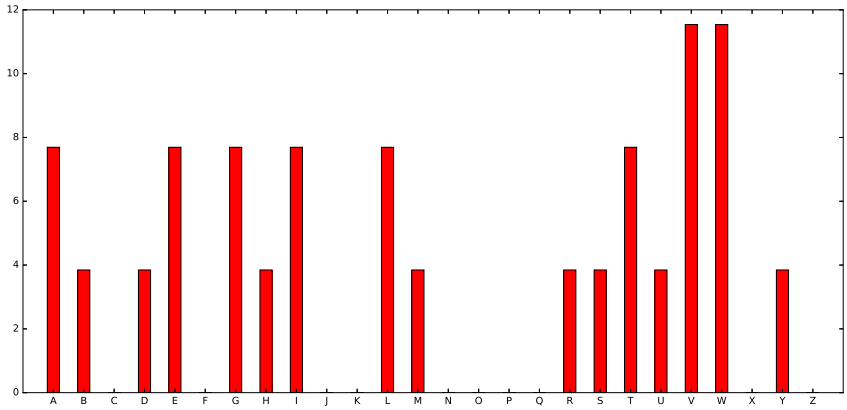


FIGURE 14.3 – vigen1.eps

Ce chiffre, qui a résisté trois siècles aux cryptanalystes, est pourtant relativement facile à casser, grâce à une méthode mise au point indépendamment par Babagge et Kasiski. Une autre méthode complètement différente a été encore mise au point plus tard par le commandant Bazeries.
Comment déchiffrer Vigenère sans connaître la clé ?
Les techniques les plus courantes utilisent des méthodes statistiques qui permettent de retrouver la longueur de la clé, puis une analyse des fréquences permet de retrouver la clé.

Test de Kasiski

Le test de Kasiski consiste à repérer des répétitions de lettres dans le texte chiffré.
ABC apparaît trois fois dans le message *ABCXYZABCKLMNOPQRSABC*.
Le fait qu'il existe une répétition signifie soit qu'une même suite de lettres du texte clair a été chiffrée avec une même partie de la clef, soit que différentes suites de lettres du texte en clair se chiffrent de la même façon. Cette seconde possibilité n'a qu'une faible probabilité d'arriver.
En analysant les écarts entre deux redondances de séquences identiques, nous pouvons déterminer un multiple de la longueur de la clé. En analysant pour chaque écart, les diviseurs possibles, nous pouvons déduire avec une grande probabilité la longueur de la clé.
Les positions de *ABC* sont 0, 6 et 18, les écarts sont donc de 6, 12 et 18, les diviseurs les plus courants de ces nombres sont 2, 3 et 6, la clé a donc une forte probabilité d'être de longueur 2, 3 ou 6.

Test de l'indice de coïncidence

Le test de l'indice de coïncidence consiste à prendre une lettre sur n dans le message, et de mesurer l'indice de coïncidence. Plus il est élevé, plus la probabilité que n soit la longueur de la clé est grande. En effet, prendre une lettre sur n, lorsque n est la longueur de la clé, revient à prendre une série de lettres chiffrées toujours chiffrées avec le même décalage, l'indice de coïncidence est donc égal à celui du texte clair.

⇒ **Activité 14. 86**

Écrivez une fonction `texte_decale(texte, dec)` (quasiment identique à `cesar_chiffre_texte(texte, k)`) qui décale une chaîne `texte` avec un décalage de `dec`, entier relatif.

Le programme :

⇒ **Activité 14. 87**

Écrivez une fonction `indice_corr_max(texte)` qui détermine l'indice de coïncidence maximum pour la langue française.

Le programme :

Voici par exemple ci-dessous le contenu du fichier `poeme2_code.txt` chiffré par la méthode de Vigenère.

EVMC	DKWL	PSSG	DNFM	RRKT	AOWK	JEIE	OKLM	DLZT	RMWX	TZRS
RXWV	HLMI	EJMM	AHYT	CAGA	TUIY	OEAY	JVPF	UXUP	DJIS	ELAU
ECIF	UXDY	JVGW	OLWL	TSIP	UJMM	AHYT	CAGA	TUYI	IEWX	DLVA
OBKM	PLTA	AVWZ	TEWJ	IMWT	PKSX	LXUW	CKVT	UGSZ	QIIS	AGKC
CAEG	DBFL	PEWJ	NUGQ	HFYS	AGKC	CVJD	RXLA	TTER	HXJL	TIVX
EKWT	PIFG	ELSV	HIMT	NWAZ	TJEC	SUGC	VVVE	AKXW	XJPD	ILWI
JRVG	IOWD	XKIB	ABKQ	AGIJ	TTMA	HZFX	EGEM	IKVT	DXDW	CXYT
STFV	TVWP	VTFB	SVWT	DXUQ	SVVC	EISA	HVHT	CHMZ	PXIG	AMLM
CUVT	AMLM	CUVT	SBDT	TWEJ	TIWV	SRRI	DXKI	CEIT	SESD	XKIH
SXGC	ARPT	NMWC	GUIA	AKJQ	KVIS	EEGQ	HVEJ	NTQI	CKEJ	CNFZ
PGTD	RMSD	TTPP	RXMA	HZXT	DNLI	QCIP	UJMI	CUPD	ILWI	JRVG
IOWA	XCEG	RBNM	DSWT	ROWZ	AVTA	ULHZ	DWSC	DLAT	TEGT	AMLM
CUVT	QNWT	DZWT	ANWV	IIIS	AGKT	PTEV	EXLY	JRRS	IEWA	IVRI
RXXM	GDIG	DHMK	TDIC	TESX	DIXT	AOWK	AVTX	NVWI	JGYX	SXXN
PTIG	UGSC	CCIH	BTJZ	TRYM	EMSG	PEXH	OBFL	TEII	ONUP	TIEJ
CNFM	SVWE	LNEM	HUIA	OBKM	PLJP	IKWM	CJYX	TXDM	EFVI	RTAB
SVPP	RUJM	TEGW	OBKQ	HJEC	TESX	ALWQ	EEDM	SVWT	SUJI	CTLT
SIGC	GCSX	SXSC	EVMC	DKWI	JJWX	LXNM	GKJT	UBDT	PXIT	TESN
GRMR	HXMZ	SLZT	NMDI	EFYH	SBWZ	TUYH	OEWQ	AVXA	EUJC	XKHT
SUWB	TJHT	LAWZ	QVHP	NLDI	RYEA	ENJL	TCII	EXLX	JZWP	TMWV
SIIF	UXDW	XJIP	ULWL	TTMS	ETUP	PEXT	RLAT	DZWT	ANFM	RYEC
TXHI	HTIH	TFSC	KRMH	SBYV	TDEX	SLAT	RYEC	TXUM	HKFD	NLAO
CVWX	GGWY	JVZD	ULHW	JMIO	SBYV	TIEA	OKKD	DLWP	RKSK	WVDI
ONLL	DLGT	MXFB	JEIS	ELHT	JDIH	DXDW	XJIP	UXLD	DLWT	CKAD
TQZD	TKWV	DDHP	NLMV	RFMC	DNLI	QCIP	UCSK	FLIH	PKWD	TIX

Pour déchiffrer un message codé par la méthode de Vigenère, il est nécessaire de trouver la longueur de la clé utilisée pour le codage.

Pour cela, nous recherchons des motifs dans le texte codé, c'est-à-dire des suites de caractères identiques apparaissant plusieurs fois dans le message. La longueur de la clé a de bonnes chances d'être un diviseur du nombre de lettres de la chaîne située entre le début du premier motif et celui du second motif. Par exemple, il y a 56 caractères entre les deux occurrences du motif `JMM AHYT CAGA TU` dans le message codé, $54 \times 4 = 216$ caractères entre les deux occurrences `PD ILWI JRVG IOW` et 24 caractères entre les deux occurrences `UGC`. Il est probable que la clé soit de longueur égale à $pgcd\ (56, 216, 24)$, soit 8.

Il y a 8 caractères entre les deux occurrences du motif `AMLM CUVT` dans le message codé, ce qui semble confirmer la longueur de la clé.

Il faut donc chercher dans le texte crypté toutes les séquences de 3 ou 4 caractères (ou plus) qui se répètent deux fois. La distance entre ces séquences est probablement (mais ce n'est pas sûr) un multiple de la taille de la clé.

Vous pourrez construire une liste des distances entre deux chaînes répétées. Ensuite à l'aide d'une boucle, vous balaierez les longueur de clé (entre 3 et 10 par exemple) en testant si la distance est un multiple de la longueur de la clé. Vous choisirez la longueur qui a le meilleur score.

⇒ **Activité 14. 88**

Écrivez une fonction `longueur_clef(texte)` qui détermine la longueur de la clé d'une chaîne `texte`. Vous pourrez utiliser la méthode `find()` dont le comportement est illustré ci-dessous :

```
>>> chain = "abcjkhgsjhgabcjjdjkabc"
>>> chain.find("abc")
0
>>> chain.find("jkh")
3
```

Le programme :

Lorsque la longueur n de la clé est connue, il faut trouver le mot clé. On commence par découper le texte T en n sous textes : T_0, T_1, ..., T_{n-1}.

$$T_i = T[i], T[i+n], T[i+2n], \ldots$$

La particularité des textes T_i est qu'ils sont codés avec le même alphabet (qui correspond a un décalage de k_i de l'alphabet). Trouvez la clé est équivalent à trouver les décalages k_0, k_1, ..., k_{n-1}.

- Une première méthode (simple) consiste a calculer pour chacun des textes la lettre la plus fréquente (on suppose alors que c'est une lettre E et nous en déduisons le décalage correspondant). Un défaut de cette méthode est que si le texte est trop court ou si la clé est trop longue, on ne peut pas assurer que cela soit toujours la lettre E qui soit la lettre la plus fréquente.

- Nous allons utiliser une deuxième méthode :
  Pour cela nous allons calculer successivement :

  $$d_1 = k_1 - k_0, d_2 = k_2 - k_0, \ldots, d_{n-1} = k_{n-1} - k_0$$

  Attention, ce point est un peu délicat à comprendre : Pour calculer d_i, nous allons boucler sur tous les décalages possible, de 0 à 25. La partie d'algorithme peut alors être :

Décalage

```
DEBUT_ALGORITHME
   indice ← 0
   decalage ← 0
   POUR d ALLANT_DE 0 A 25
      DEBUT_POUR
      t ← concaténation de T_0 avec T_i auquel on a appliqué le décalage d
      Calcul de l'indice de coïncidence du texte t
      SI I_C(t) > indice ALORS
         DEBUT_SI
           indice ← I_C(t)
           decalage ← d
           FIN_SI
        FIN_POUR
     di ← decalage
FIN_ALGORITHME
```

En effet quand nous trouvons le bon décalage alors cela veut dire que les deux textes T_0 et T_i sont codés avec exactement le même alphabet et correspondent donc a du français : l'indice de coïncidence est alors maximal.
Il ne reste plus maintenant qu'à fixer l'origine pour cela nous concaténons tous les textes T_i, chacun avec son décalage d_i et nous cherchons la lettre la plus fréquente qui va correspondre alors à la lettre E, ce qui nous permet de fixer le décalage global.

⇒ **Activité 14. 89**

Écrivez une fonction `recherche_clef(texte, long_clef)` qui retourne la clé d'une chaîne `texte` grâce à la longueur `long_clef`.
Vous pourrez utiliser un dictionnaire ainsi que la méthode `keys()` dont le comportement est illustré dans le programme ci-dessous :

dico-01.py

```
# -*- coding: utf-8 -*-

dico = {}
dico["a"] = 0
dico["b"] = 1
dico["c"] = 2
dico["d"] = 3
dico["début"] = 6
print("dico 1 : ", dico)
dico["a"] = dico["a"] + 11
print("dico 2 : ",dico)
maxi = 0
for car in dico.keys():
    if dico[car] > maxi:
        print("début : ", maxi)
        maxi = dico[car]
        print("fin : ", maxi)
print("finalement : ", maxi)
```

dont le résultat est :

```
dico 1 :  {'b': 1, 'début': 6, 'c': 2, 'd': 3, 'a': 0}
dico 2 :  {'b': 1, 'début': 6, 'c': 2, 'd': 3, 'a': 11}
début :  0
fin :  1
début :  1
fin :  6
début :  6
fin :  11
finalement :  11
```

La fonction `recherche_clef(texte, long_clef)` :

14.6.2.3 Decryptage sans la clé

⇒ **Activité 14. 90**

Il reste ensuite à casser le message grâce à la procédure `casse_texte(texte)`, à écrire bien sûr... Celle-ci fait appel bien entendu à `longueur_clef(texte)` et `recherche_clef(texte, long_clef)`.

La procédure `casse_texte(texte)` :

14.7 Chiffrement RSA

Pour crypter un message, nous commençons par le transformer en un – ou plusieurs – nombres. Les processus de chiffrement et déchiffrement font appel à plusieurs notions :

- On choisit deux **nombres premiers** p et q que l'on garde secrets et on pose $n = p \times q$. Le principe étant que même connaissant n il est très difficile de retrouver p et q (qui sont des nombres ayant des centaines de chiffres).
- La clé secrète et la clé publique se calculent à l'aide de l'**algorithme d'Euclide** et des **cœfficients de Bézout**.
- Les calculs de cryptage se feront **modulo** n.
- Le déchiffrement fonctionne grâce à une variante du **petit théorème de Fermat**.

Dans cette section, c'est Célestin qui veut envoyer un message secret à Sidonie. La processus se décompose ainsi :

1. Sidonie prépare une clé publique et une clé privée,
2. Célestin utilise la clé publique de Sidonie pour crypter son message,
3. Sidonie reçoit le message crypté et le déchiffre grâce à sa clé privée.

14.7.1 Calcul de la clé publique et de la clé privée

14.7.1.1 Choix de deux nombres premiers

Sidonie effectue, une fois pour toute, les opérations suivantes (en secret) :

- elle choisit deux nombres premiers distincts p et q (dans la pratique ce sont de très grand nombres, jusqu'à des centaines de chiffres),
- Elle calcule $n = p \times q$,
- Elle calcule $\varphi(n) = (p-1) \times (q-1)$.

Exemple 1

- $p = 5$ et $q = 17$
- $n = p \times q = 85$
- $\varphi(n) = (p-1) \times (q-1) = 64$

Vous noterez que le calcul de $\varphi(n)$ n'est possible que si la décomposition de n sous la forme $p \times q$ est connue. D'où le caractère secret de $\varphi(n)$ même si n est connu de tous.

Exemple 2

- $p = 101$ et $q = 103$
- $n = p \times q = 10\,403$
- $\varphi(n) = (p-1) \times (q-1) = 10\,200$

14.7.1.2 Choix d'un exposant et calcul de son inverse

Sidonie continue :

- elle choisit un exposant e tel que $\text{pgcd}(e, \varphi(n)) = 1$,
- elle calcule l'inverse d de e module $\varphi(n)$: $d \times e \equiv 1 \pmod{\varphi(n)}$. Ce calcul se fait par l'algorithme d'Euclide étendu.

Exemple 1

- Sidonie choisit par exemple $e = 5$ et on a bien $\text{pgcd}(e, \varphi(n)) = \text{pgcd}(5, 64) = 1$,
- Sidonie applique l'algorithme d'Euclide étendu pour calculer les cœfficients de Bézout correspondant à $\text{pgcd}(e, \varphi(n)) = 1$. Elle trouve $5 \times 13 + 64 \times (-1) = 1$. Donc $5 \times 13 \equiv 1 \pmod{64}$ et l'inverse de e modulo $\varphi(n)$ est $d = 13$.

Exemple 2

- Sidonie choisit par exemple $e = 7$ et on a bien $\text{pgcd}(e, \varphi(n)) = \text{pgcd}(7, 10\,200) = 1$,
- L'algorithme d'Euclide étendu pour $\text{pgcd}(e, \varphi(n)) = 1$ donne $7 \times (-1457) + 10 \times 200 \times 1 = 1$. Mais $-1457 \equiv 8743 \pmod{\varphi(n)}$, donc pour $d = 8743$, on a $d \times e \equiv 1 \pmod{\varphi(n)}$.

14.7.1.3 Clé publique

La **clé publique** de Sidonie est constituée des deux nombres : n et e.

Et comme son nom l'indique, Sidonie communique sa clé publique au monde entier.

Exemple 1

$n = 85$ et $e = 5$.

Exemple 2

$n = 10\,403$ et $e = 7$.

14.7.1.4 Clé privée

Sidonie garde pour elle sa **clé privée** : d

Sidonie détruit en secret p, q et $\varphi(n)$ qui ne sont plus utiles. Elle conserve secrètement sa clé privée.

Exemple 1

$d = 13$

Exemple 2

$d = 8743$

14.7.2 Chiffrement du message

Célestin veut envoyer un message secret à Sidonie. Il se débrouille pour que son message soit un entier (quitte à découper son texte en bloc et à transformer chaque bloc en un entier).

14.7.2.1 Message

Le message est un entier m, tel que $0 \leq m < n$.

Exemple 1

Célestin veut envoyer le message $m = 10$.

Exemple 2

Célestin veut envoyer le message $m = 1234$.

14.7.2.2 Message chiffré

Célestin récupère la clé publique de Sidonieé : n et e avec laquelle il calcule, à l'aide de l'algorithme d'exponentiation rapide, le message chiffré :

$$x \equiv m^e \pmod{n}$$

Il transmet ce message x à Sidonie.

Exemple 1

$m = 10$, $n = 85$ et $e = 5$ donc

$$x \equiv m^e \pmod{n} \equiv 10^5 \pmod{85}$$

On peut ici faire les calculs à la main :

$$10^2 \equiv 100 \equiv 15 \pmod{85}$$
$$10^4 \equiv (10^2)^2 \equiv 15^2 \equiv 225 \equiv 55 \pmod{85}$$
$$x \equiv 10^5 \equiv 10^4 \times 10 \equiv 55 \times 10 \equiv 550 \equiv 40 \pmod{85}$$

Le message chiffré est donc $x = 40$.

Exemple 2

$m = 1234$, $n = 10\,403$ et $e = 7$ donc :

$$x \equiv m^e \pmod{n} \equiv 1234^7 \pmod{10\,403}$$

On utilise l'ordinateur pour obtenir que $x = 10\,378$.

14.7.3 Déchiffrement du message

Sidonie reçoit le message x chiffré par Célestin, elle le décrypte à l'aide de sa clé privée d, par l'opération : $m \equiv x^d \pmod{n}$ qui utilise également l'algorithme d'exponentiation rapide.

Nous allons prouver dans le lemme 14.7.5, que par cette opération Sidonie retrouve bien le message original m de Célestin.

Exemple 1

$c = 40$, $d = 13$, $n = 85$ donc

$$x^d \equiv (40)^{13} \pmod{85}$$

Calculons à la main $40^{13} \equiv \pmod{85}$ on note que $13 = 8 + 4 + 1$, donc $40^{13} = 40^8 \times 40^4 \times 40$.

$$\begin{aligned} 40^2 &\equiv 1600 \equiv 70 \pmod{85} \\ 40^4 &\equiv (40^2)^2 \equiv 70^2 \equiv 4900 \equiv 55 \pmod{85} \\ 40^8 &\equiv (40^4)^2 \equiv 55^2 \equiv 3025 \equiv 50 \pmod{85} \end{aligned}$$

Donc

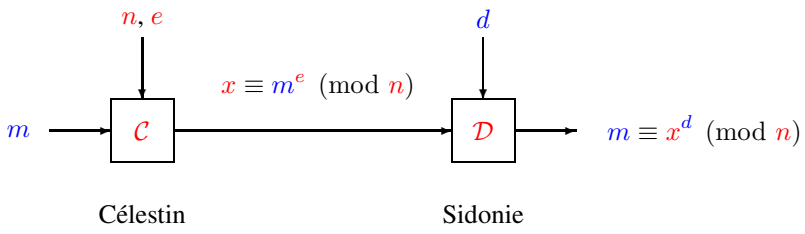
$$x^d \equiv 40^{13} \equiv 40^8 \times 40^4 \times 40 \equiv 50 \times 55 \times 40 \equiv 10 \pmod{85}$$

qui est bien le message m de Célestin.

Exemple 2

$c = 10\,378$, $d = 8743$, $n = 10\,403$. On calcule par ordinateur $x^d \equiv (10\,378)^{8743} \pmod{10\,403}$ qui vaut exactement le message original de Célestin $m = 1234$.

14.7.4 Schéma



Clés de Sidonie :

- publique : n, e
- privéeé : d

14.7.5 Lemme de déchiffrement

Le principe de déchiffrement repose sur le petit théorème de Fermat amélioré.

Lemme

Soit d l'inverse de e modulo $\varphi(n)$. Si $x \equiv m^e \pmod{n}$ alors $m \equiv x^d \pmod{n}$.

Ce lemme prouve bien que le message original m de Célestin, chiffré par clé publique de Sidonie (e,n) en le message x, peut-être retrouvé par Sidonie à l'aide de sa clé secrète d.

Démonstration

- Que d soit l'inverse de e modulo $\varphi(n)$ signifie $d\cdot e \equiv 1 \pmod{\varphi(n)}$. Autrement dit, il existe $k\in\mathbb{Z}$ tel que $d\cdot e = 1 + k\,\varphi(n)$.
- On rappelle que par le petit théorème de Fermat généralisé : lorsque m et n sont premiers entre eux
$$m^{\varphi(n)} \equiv m^{(p-1)(q-1)} \equiv 1 \pmod{n}$$
- **Premier cas** $\mathrm{pgcd(m,n)} = 1$.
Notons $c \equiv m^e \pmod{n}$ et calculons x^d :
$$x^d \equiv (m^e)^d \equiv m^{e\,d} \equiv m^{1+k\,\varphi(n)} \equiv m\,m^{k\,\varphi(n)} \equiv m\,(m^{\varphi(n)})^k \equiv m\,(1)^k \equiv m \pmod{n}$$
- **Deuxième cas** $\mathrm{pgcd(m,n)} \neq 1$.
Comme n est le produit des deux nombres premiers p et q et que m est strictement plus petit que n alors si m et n ne sont pas premiers entre eux cela implique que p divise m ou bien q divise m (mais pas les deux en même temps). Faisons l'hypothèse $\mathrm{pgcd(m,n)} = \mathrm{p}$ et $\mathrm{pgcd(m,q)} = 1$, le cas $\mathrm{pgcd(m,n)} = \mathrm{q}$ et $\mathrm{pgcd(m,p)} = 1$ se traiterait de la même manière.
Étudions $(m^e)^d$ à la fois modulo p et modulo q à l'image de ce que nous avons fait dans la preuve du théorème de Fermat amélioré.
  - modulo p : $m \equiv 0 \pmod{p}$ et $(m^e)^d \equiv 0 \pmod{p}$ donc $(m^e)^d \equiv m \pmod{p}$,
  - modulo q : $(m^e)^d \equiv m\times(m^{\varphi(n)})^k \equiv m\times(m^{q-1})^{(p-1)k} \equiv m \pmod{q}$.

  Comme p et q sont deux nombres premiers distincts, ils sont premiers entre eux et on peut écrire comme dans la preuve du petit théorème de Fermat amélioré que
$$(m^e)^d \equiv m \pmod{n}$$

14.7.6 Algorithmes

La mise en œuvre est maintenant très simple. Sidonie choisit deux nombres premiers p et q et un exposant e.

Voici le calcul de la clé secrète :

rsa-tex1.py

```
1  def cle_privee(p,q,e) :
2      n = p * q
3      phi = (p-1)*(q-1)
4      c,d,dd = euclide_etendu(e,phi)          # Pgcd et coeff de Bézout
5      return(d % phi)                         # Bon représentant
```

Le chiffrement d'un message m est possible par tout le monde, connaissant la clé publique (n,e).

rsa-tex2.py

```
1  def codage_rsa(m,n,e):
2      return pow(m,e,n)
```

Seule Sidonie peut déchiffrer le message crypté x, à l'aide de sa clé privée d.

rsa-tex3.py

```
1 def decodage_rsa(x,n,d):
2     return pow(x,d,n)
```

Lettre	Français	Anglais	Allemand	Espagnol	Portugais	Italien
a	7.636	8.167	6.516	11.525	14.634	11.745
b	0.901	1.492	1.886	2.215	1.043	0.927
c	3.260	2.782	2.732	4.019	3.882	4.501
d	3.669	4.253	5.076	5.010	4.992	3.736
e	14.715	12.702	16.396	12.181	12.570	11.792
f	1.066	2.228	1.656	0.692	1.023	1.153
g	0.866	2.015	3.009	1.768	1.303	1.644
h	0.737	6.094	4.577	0.703	0.781	0.636
i	7.529	6.966	6.550	6.247	6.186	10.143
j	0.613	0.153	0.268	0.493	0.397	0.011
k	0.049	0.772	1.417	0.011	0.015	0.009
l	5.456	4.025	3.437	4.967	2.779	6.510
m	2.968	2.406	2.534	3.157	4.738	2.512
n	7.095	6.749	9.776	6.712	4.446	6.883
o	5.796	7.507	2.594	8.683	9.735	9.832
p	2.521	1.929	0.670	2.510	2.523	3.056
q	1.362	0.095	0.018	0.877	1.204	0.505
r	6.693	5.987	7.000	6.871	6.530	6.367
s	7.948	6.327	7.270	7.977	6.805	4.981
t	7.244	9.056	6.154	4.632	4.336	5.623
u	6.311	2.758	4.166	2.927	3.639	3.011
v	1.838	0.978	0.846	1.138	1.575	2.097
w	0.074	2.360	1.921	0.017	0.037	0.033
x	0.427	0.150	0.034	0.215	0.253	0.003
y	0.128	1.974	0.039	1.008	0.006	0.020
z	0.326	0.074	1.134	0.467	0.470	1.181
à	0.486	0	0	0	0.072	0.635
â	0.051	0	0	0	0.562	0
á	0	0	0	0.502	0.118	0
ä	0	0	0.578	0	0	0
ã	0	0	0	0	0.733	0
œ	0.018	0	0	0	0	0
ç	0.085	0	0	0	0.530	0
è	0.271	0	0	0	0	0.263
é	1.504	0	0	0.433	0.337	0
ê	0.218	0	0	0	0.450	0
ë	0.008	0	0	0	0	0
î	0.045	0	0	0	0	0
ï	0.005	0	0	0	0	0
í	0	0	0	0.725	0.132	0
ì	0	0	0	0	0	0.030
ñ	0	0	0	0.31	0	0
ß	0	0	0.307	0	0	0
ô	0.023	0	0	0	0.635	0
ò	0	0	0	0	0	0.002
ó	0	0	0	0.827	0.296	0
ö	0	0	0.443	0	0	0
ú	0	0	0	0.168	0.207	0
ù	0.058	0	0	0	0	0.166
ü	0	0	0.995	0.012	0.026	0

TABLE 14.2 – Fréquences en pourcentages (https://en.wikipedia.org/wiki/Letter_frequency)

15

Bases de données

Ce chapitre constitue un rappel de première année et ne peut bien évidemment remplacer le cours...
Il est très largement inspiré du cours disponible ici : http://michel.stainer.pagesperso-orange.fr/

15.1 Introduction

Les données constituant une base de données sont organisées en tables. Une table contient des enregistrements composés de différents champs appelés également attributs. Le type associé au champ définit le domaine dans lequel le champ pourra prendre ses valeurs : par exemple entier, flottant, chaîne, date, ...
Les chaînes s'écrivent entre 'simples' ou "doubles" quotes (apostrophes ou guillemets).
Chaque table est désignée de manière unique par un identificateur (son nom) au sein de la base de données, de même que chaque champ au sein d'une table.
On peut représenter une table par un tableau dont les colonnes correspondent aux champs et les lignes aux enregistrements.
Une clé primaire d'une table est un champ (ou un ensemble de champs) qui identifie de manière unique chaque enregistrement dans la table. Chaque table devrait avoir une clé primaire.
Une clé étrangère dans une table est un champ (ou un ensemble de champs) qui fait référence à un champ (ou un ensemble de champs) d'une autre table (généralement la clé primaire).
Les requêtes SQL permettent d'interroger la base de données.
Pour ce faire, on peut utiliser des logiciels comme *SQLiteStudio* ou *Sqliteman*.

15.2 Création d'une base de données à partir de fichiers .csv

La base de données utilisée plus loin a été créée à partir de fichiers `.csv` à l'aide des commandes suivantes qui permettent d'importer des tables (dans le terminal Linux) : ici on importe dans la base `bdd_geo.db` les trois tables `communes`, `departements` et `regions` à partir des fichiers `villes_france_simple.csv`, `depts.csv` et `regs.csv`.

```
ordi@ordi-All-Series:~:$ cd repertoire
ordi@ordi-All-Series:~:repertoire$ sqlite3 bdd_geo.db
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .mode csv
sqlite> .import villes_france_simple.csv communes
sqlite> .import depts.csv departements
sqlite> .import regs.csv regions
sqlite> .exit
ordi@ordi-All-Series:/repertoire$
```

15.3 Connexion à base via *Python*

On peut se connecter via *Python* à une base de données via le module *sqlite3*.
L'exécution d'une requête renvoie alors un itérateur de type `Cursor`, que l'on peut utiliser directement à l'aide d'une boucle `for`, ce qui permet de parcourir les lignes du résultat de la requête.
Ces lignes se présentent sous forme de tuples. On peut aussi convertir le résultat en "liste de lignes" grâce à la méthode `fetchall()`. On peut enfin obtenir une ligne après l'autre avec la méthode `fetchone()` C'est ceete dernière méthode que l'on utilisera lorsque la requête n'attend qu'un seul résultat.

⚠ Ces trois procédés détruisent les lignes utilisées : si on souhaite les utiliser plusieurs fois, il est nécessaire de les stocker dans une liste par une instruction du type `result = cursor.fetchall()`.

```
import sqlite3

db_loc = sqlite3.connect('bdd_geo.db')
cursor = db_loc.cursor()

cursor.execute('''SELECT ... FROM ...''')
exemple1 = cursor.fetchone() # première ligne de la recherche
print(exemple1)

cursor.execute('''SELECT ... FROM ...''')
result = cursor.fetchall() # pour sélectionner tous les élements
print(result)
for el in result:
    print(el)
```

15.4 Requêtes

15.4.1 Définitions

1. **Valeur** `NULL`
   Un champ qui n'est pas renseigné, donc vide, contient la valeur `NULL`. Cette valeur est différente de zéro et représente l'absence de valeur.
2. **Expressions**
   Les expressions valides mettent en jeu des noms de champs, le mot clé "*" (qui signifie "tous les champs"), des constantes, des fonctions et des opérateurs classiques. Il existe des fonctions logiques, mathématiques, de manipulation de chaînes, de dates et des fonctions d'agrégation. Les fonctions d'agrégation permettent de calculer un résultat numérique (un entier ou un flottant) à partir d'un ensemble de valeurs. Les fonctions d'agrégation sont au nombre de cinq : `COUNT`, `SUM`, `MIN`, `MAX` et `AVG`.

15.4.2 Interrogation d'une base

Dans les descriptions qui suivent, les expressions entre crochets sont facultatives.
Lorsque plusieurs opérateurs sont utilisables à un endroit donné, ils sont séparés par des barres verticales (il faut en mettre un seul dans la requête).
La syntaxe classique est la suivante :

```
SELECT ... FROM ... [WHERE ...] [GROUP BY ... [HAVING ...]] [ORDER BY ...] [LIMIT
    ... [OFFSET ...]]
```

1. `SELECT ...`
   La commande

```
SELECT [DISTINCT] expr1 [[AS] alias1], expr2 [[AS] alias2], ...
```

réalise une projection.
`expr1`, `expr2`, ... indiquent quelles expressions devront être renvoyées, par exemple des noms de champs.
S'il y a une ambiguïté (cas de deux tables contenant des champs de même nom), il est nécessaire de préfixer le nom du champ par celui de la table suivi d'un point.
Le mot clé `AS` permet le renommage : ainsi, `alias1`, `alias2`, ... sont des alias qui fournissent un identificateur abrégé pour chaque champ concerné, pour la suite de la requête. En outre, un tel alias est utilisé comme titre de la colonne correspondante dans le résultat de la requête (le mot clé `AS` est facultatif mais il améliore grandement la lisibilité).
Le mot clé `DISTINCT` permet de supprimer les doublons.

2. `FROM ...`
   La commande

```
FROM table1 [[AS] alias1], table2 [[AS] alias2], ...
```

   donne la liste des tables participant à l'interrogation. `alias1`, `alias2`, ... sont des alias attribués aux tables pour le temps de la requête. Quand une table se voit attribuer un alias, elle n'est alors plus reconnue sous son nom d'origine dans la requête.

3. `WHERE ...`
   La commande `WHERE` permet d'effectuer une restriction dans la recherche effectuée dans une table ou un produit cartésien de tables.

   (a) Prédicats simples
   Un prédicat simple est la comparaison de plusieurs expressions au moyen d'un opérateur logique `opérateur_logique` :

```
WHERE expr1 opérateur\_logique expr2
```

   pour les opérateurs classiques.

```
WHERE expr1 [NOT] LIKE expr2
```

   où `expr2` est un masque, une chaîne de caractères contenant au moins un joker : _ pour un unique caractère ou % pour un nombre quelconque de caractères. Un masque comme '%7%' permet de filtrer les chaînes, mais aussi les nombres contenant un 7.

```
WHERE expr1 IS [NOT] NULL
```

   La valeur spéciale NULL est adaptée pour un champ vide.

```
WHERE expr1 [NOT] IN (expr2,expr3, ...
```

   teste l'appartenance à la liste d'expressions.

```
WHERE [NOT] EXISTS (expr1)
```

   teste si la sous-requête `expr1` renvoie au moins un résultat.

   (a) Prédicats composés
   Les opérateurs logiques `AND` et `OR` permettent de combiner plusieurs prédicats. `AND` est prioritaire par rapport à `OR` mais l'utilisation de parenthèses permet de modifier l'ordre d'évaluation.

(b) Sous-requêtes
Le critère de recherche employé dans une clause `WHERE` (l'expression à droite d'un opérateur de comparaison) peut être le résultat d'un `SELECT`.
Dans le cas des opérateurs classiques, la sous—interrogation ne doit ramener qu'une ligne et une colonne. Elle peut ramener plusieurs lignes à la suite des opérateurs `[NOT] IN`, `[NOT] EXISTS`, ou encore à la suite des opérateurs classiques `opérateur_logique` moyennant l'ajout d'un mot clé (`ALL` ou `ANY`).

```
WHERE expr1 opérateur_logique ALL (SELECT ...)
WHERE expr1 opérateur_logique ANY (SELECT ...)
```

- `ALL` : la comparaison est vraie si elle est vraie pour tous les éléments de l'ensemble (donc vraie si l'ensemble des enregistrements est vide).
- `ANY` : la comparaison est vraie si elle est vraie pour au moins un élément de l'ensemble (donc fausse si l'ensemble des enregistrements est vide). Vous pourrez que noter que `= ANY (...)` équivaut à `IN (...)`.

4. `GROUP BY ... [HAVING ...]`
La commande

```
GROUP BY expr1, expr2, ...
```

permet de subdiviser la table en groupes, chaque groupe étant l'ensemble des enregistrements ayant une valeur commune pour les expressions spécifiées. Les champs de la clause `SELECT` doivent alors être des fonctions d'agrégation ou des expressions figurant dans la clause `GROUP BY`. Comme son nom l'indique, une fonction d'agrégation s'applique à chaque groupe d'enregistrements créé par la clause `GROUP BY`.
`HAVING prédicat` sert à sélectionner une partie seulement des groupes formés par `GROUP BY`. Le prédicat ne peut porter que sur des fonctions d'agrégation ou des expressions figurant dans la clause `GROUP BY`.

5. `ORDER BY ...`
La commande

```
ORDER BY expr1 [ASC | DESC], expr2 [ASC | DESC], ...
```

classe les enregistrements retournés selon les valeurs de `expr1` puis `expr2`, ...
L'ordre par défaut est croissant, la clause `ASC` est sous-entendue et par conséquent facultative. La clause `DESC` implique un tri par ordre décroissant. Pour préciser lors d'un tri sur quelle expression va porter le tri, il est possible de donner sa position dans la liste des expressions de la clause `SELECT` ou encore l'alias qu'on lui a attribué.

6. `LIMIT ...`
La commande

```
LIMIT nb [OFFSET dec]
```

limite à `nb` le nombre d'enregistrements retournés. Cette clause est souvent utilisée après un tri.
L'option `OFFSET dec` permet d'occulter les `dec` premiers résultats. Ainsi par exemple, `LIMIT 3 OFFSET 8` retournera les enregistrements compris entre 8 et 11.

15.4.3 Jointures

Avec une base de données complexe, on a souvent besoin de rassembler des données réparties dans plusieurs tables, typiquement concaténer les enregistrements provenant de deux tables, `table1` et `table2`, et vérifiant `table1.champ1 = table2.champ2` (l'un des deux `champ1` ou `champ2` est souvent une clé primaire dans sa table).
Pour ce faire, une solution (coûteuse) consiste à former le produit cartésien des deux tables, puis à sélectionner parmi le (trop) grand nombre d'enregistrements obtenus :

```
SELECT expr1, expr2, ... FROM table1, table2 WHERE table1.champ1 = table2.champ2
```

Il faut préférer à ce type de requête l'utilisation d'une jointure (partie du produit cartésien formée des enregistrements vérifiant la condition). Une telle jointure est crée par les logiciels de gestion de bases de données de façon beaucoup plus efficace que la sélection dans le produit cartésien. La syntaxe est alors la suivante :

```
SELECT expr1, expr2, ... FROM table1 JOIN table2 ON table1.champ1 = table2.champ2
```

Remarque

Le résultat de `JOIN ... ON ...` étant une nouvelle table, on peut enchaîner plusieurs clauses `JOIN`.

15.4.4 Les opérateurs ensemblistes

Les opérateurs ensemblistes `opérateur` tels que `UNION`, `INTERSECT`, `MINUS` utilisés ainsi :

```
requête1 opérateur requête2 ...
```

permettent de réaliser des opérations ensemblistes sur les résultats de plusieurs interrogations.
Les champs renvoyés par les requêtes impliquées doivent être identiques. Les opérations sont évaluées de gauche à droite mais l'utilisation de parenthèses permet de modifier cet ordre.

15.5 Contenu de la base

Sur le site http://sql.sh/736-base-donnees-villes-francaises, on peut trouver une table des communes de France. Dans le fichier *bdd_geo.db* fourni, seuls certains attributs ont été conservés. Vous y trouverez 3 tables. La première, intitulée *communes*, contient les attributs suivants :

- **id** : identifiant,
- **num_departement** : numéro du département,
- **nom** : nom de la commune,
- **canton** : numéro de canton de la commune,
- **code_postal** : code postal de la commune,
- **population_2010** : nombre d'habitants lors du recensement de 2010,
- **population_1999** : nombre d'habitants lors du recensement de 1999,
- **densite_2010** : densité en 2010,
- **surface** : surface de la commune en km^2,
- **longitude_deg** : longitude du centre de la commune en degrés,
- **latitude_deg** : latitude du centre de la commune en degrés,
- **z_min** : hauteur minimale de la commune par rapport au niveau de la mer,
- **z_max** : hauteur maximale de la commune par rapport au niveau de la mer.

La deuxième, intitulée *departements*, contient les attributs suivants :

- **num_dept** : numéro du département,
- **num_region** : numéro de la région
- **nom_dept** : nom du département

La troisième, intitulée *regions*, contient les attributs suivants :

- **num_region** : numéro de la région
- **nom_region** : nom de la région

15.6 Exercice

Sauf indication contraire, on travaille sur la population en 2010.

15.6.1 Requêtes sans jointure

1. Obtenez le nombre d'habitants en France (en supposant que chaque habitant est rattaché à exactement une commune) ?
2. Combien y-a-t-il de communes en France ? de communes ayant des noms différents ?
3. Combien de communes dans l'Ain (numéro de département 1) ?
4. Obtenez la liste des 5 plus petites communes de France en terme de surface.
5. Obtenez la liste des 6 communes de France les plus peuplées.
6. Quelles sont les 7 communes les plus densément peuplées de l'Ain ?
7. Obtenez la commune la plus au Nord ainsi que celle la plus au Sud en France métropolitaine.
8. Obtenez la liste des numéros des départements, et pour chaque département correspondant, le nombre de communes par ordre décroissant (on se limitera aux 9 premiers départements).
9. Obtenez la liste des numéros des départements, et pour chaque département correspondant, la population totale. Ordonnez par population totale décroissante en se limitant aux 11 premiers départements.
10. Combien de communes françaises contiennent la lettre "z" ou "Z" dans leur nom ?
11. Combien de communes françaises contiennent les six voyelles "a", "e", "i", "o", "u", "y" (en majuscule ou minuscule) dans leurs noms ?
12. Combien de communes dans l'Ille-et-Vilaine (35) contiennent les six voyelles "a", "e", "i", "o", "u", "y" (en majuscule ou minuscule) dans leurs noms ?
13. Quelles sont les 6 communes de France dont l'écart entre l'altitude la plus haute, et l'altitude la plus basse, est le plus élevé ?
14. Quelles sont les 5 communes du Morbihan (56) où la plus population a le plus augmenté entre 1999 et 2010 ? Répondez en augmentation absolue d'abord, puis en augmentation relative.

15.6.2 Jointures à deux tables

15. Obtenez la liste des départements (les noms), et pour chaque département correspondant, le nombre de communes. Vous ordonnerez par nombre de communes décroissant en vous limitant aux 12 premiers obtenus.
16. Obtenez la liste des départements des régions "Grand Est" et "Occitanie".
17. Quelle est la population en 2010 de chaque département (on souhaite les noms des départements en toutes lettres) ? Les départements seront classés par population décroissante en se limitant aux 6 premiers.
18. Quelle est la densité de population en 1999 de chaque département (on veut les noms des départements en toutes lettres) ? On se limitera aux 8 premiers départements.
19. Obtenez la liste des régions de France, ainsi que le nombre de départements composant la région, rangée par nombre de départements décroissant.
20. Obtenez la liste des départements (les noms) de plus de 1500000 habitants.
21. Donner la liste des départements (les noms) où il y a une ville de plus de 200000 habitants. On se limitera aux 7 premiers.
22. Obtenez les noms de commune de la région "Bretagne" (3) dont le nom contient les six voyelles "a", "e", "i", "o", "u", "y" (en majuscules ou minuscules) ?
23. Donnez toutes les communes ayant le même nombre d'habitants qu'une autre en vous limitant aux 10 premières.

15.6.3 Jointures à trois tables

24. Obtenez la liste des régions de France, et pour chaque région, la population totale, ainsi que le nom et le nombre d'habitants de la commune la plus peuplée.
25. Obtenez la liste des régions (avec le nom en français), avec la population totale de chaque région.
26. Obtenez la densité de population de chaque région en 1999.
27. Quelles sont les 6 communes de France dont l'écart entre l'altitude la plus haute, et l'altitude la plus basse, est le plus élevé ? Cette fois, précisez le nom du département et le nom de la région pour chaque commune.
28. Obtenez le nombre de communes en Normandie.

15.6.4 Utilisation de sous-requêtes

29. Obtenez (sans doublons) la liste des régions contenant des départements dont le nom commence par "f" ou "F" (vous devriez y trouver la Bretagne).
30. Obtenez (sans doublons) la liste des régions ne contenant aucune commune dont le nom contient les six voyelles.
31. Trouvez les communes ayant un nom identique à celui d'une autre commune (on se limitera à 10).

15.6.5 Un peu de dessin

32. Représenter les communes de métropole (avec un numéro de département inférieur ou égal à 95) par des points de couleur fonction de l'altitude du point le plus haut dans la commune (points placés en fonction des longitude et latitude).
33. Représenter par des points de couleur l'attractivité des cantons français, représentée par l'augmentation relative du nom d'habitants entre 1999 et 2010 (points placés en fonction des longitude et latitude).
34. Représenter par des points les 3000 plus grosses communes de métropole (points placés en fonction des longitude et latitude).

Partie 5

Annexes

Table des matières

16

Python : quelle place parmi les langages informatiques ?

16.1 Histoire du langage

La première version de *Python* date de 1991, développée par Guido VAN ROSSUM. Il utilise une syntaxe très légère, inspirée d'*ABC*. Le langage dispose de très peu de types de données, mais est riche en structures de données dynamiques, bien intégrées à la syntaxe.

La licence sous laquelle il est distribué se stabilise en 2001 sur une licence libre compatible avec la *GPL*[1]. Il s'agit d'un logiciel libre : vous pouvez modifier votre interpréteur *Python* et distribuer cette version modifiée. Toutefois, cette licence n'entraîne aucune contrainte quant à la licence qui régit les programmes écrits en *Python*.

Le modèle objet est unifié avec la version 2.2 : les types de base (`int`, `float`, `bool`) deviennent des objets comme les autres. Ils ont des méthodes, et il est possible d'en hériter. La dernière version de cette branche est la 2.7.4 d'avril 2013.

La version 3.0 vise à éliminer des redondances dans les fonctionnalités de *Python*. Il s'agit de diminuer le nombre de façons de réaliser une même opération, et donc la complexité du langage comme de sa machine virtuelle. Une conséquence est que la version 3.0 n'est pas compatible avec les versions précédentes : elle ne peut pas exécuter du code *Python* $2.x$. Certaines bibliothèques scientifiques n'ont pas encore fait cette transition et *Python(x,y)*, par exemple, utilise actuellement *Python* 2.7.3.

16.2 Caractéristiques du langage

Python est un *langage objet*, où tout est objet, y compris ce qui est communément appelé type de base : entiers, réels, booléens. Il permet l'héritage multiple.

Python est un langage de *haut niveau*, où nombre de structures de données dynamiques (listes et tables de hachages notamment) sont intégrées à la syntaxe du langage. De nombreuses bibliothèques fournissent des fonctions évoluées.

1. La *GNU General Public License* est la plus importante des licences sous lesquelles sont distribués les logiciels libres ; elle est écrite par la *Free Software Foundation*. La licence de *Python* est la *Python Software Foundation License* (PSFL), inspirée de la licence BSD de Berkeley

Python est un langage à *typage dynamique*. **Le développeur n'a pas à déclarer les variables avant de leur affecter une valeur : la création de la variable se fait au moment où elle est utile**. Il n'a pas non plus à déclarer son type : lors de son utilisation, les opérateurs (qui sont en fait des méthodes des opérandes) en testent le type pour vérifier s'ils sont dans leurs conditions d'utilisation.

Python est un langage à *typage fort* : lorsqu'un opérateur reçoit deux opérandes incompatibles, une exception est levée et l'opération n'est pas exécutée. Un langage à *typage faible* aurait tenté une conversion de l'un des opérandes vers un type compatible et effectué l'opération, au risque d'avoir un résultat inattendu : `4+"2"` doit-il fournir la chaîne `"42"`, l'entier `6` ou une erreur ?
Python choisit la dernière option.

Python permet la *glue logicielle*, et peut notamment appeler des fonctions présentes dans des fichiers (`.o`) issus de programmes *C*. Des *bindings* (liaisons) permettent la traduction des données du *C* en objets *Python*, et inversement.

Python est un langage à *ramasse-miette* : de l'espace mémoire est réservé pour les objets lorsqu'ils sont créés, et la machine virtuelle se charge de libérer cet espace lorsqu'elle se rend compte que l'objet ne peut plus être utilisé par le programme. Cela libère le programmeur de la gestion de la mémoire, au risque parfois d'en consommer trop (si les miettes ne sont pas identifiées comme telles) et un surcoût de temps d'exécution.

16.3 Python et les autres langages

- *C*, *Fortran*, *Ada*, *Pascal/Delphi* sont des langages compilés : un compilateur doit transformer un fichier source, généralement assez indépendant de la plate-forme matérielle, en un code exécutable, adapté au processeur et à l'architecture voulus.
- *Java* est un langage semi-compilé : un compilateur vérifie la syntaxe du fichier source et le transforme en un *byte-code* indépendant de la machine, et une *machine virtuelle*, spécifique à chaque architecture, exécute le byte-code.
- *Matlab*, *tcsh*, *bash* et les autres *shells* sont des langages interprétés : le fichier source est exécuté ligne par ligne. Il n'y a pas d'étape intermédiaire entre l'écriture du code et son exécution. Le programme qui se charge d'exécuter chacune des lignes est l'*interpréteur*.
- *Perl* est un langage interprété précompilé, dont la syntaxe est proche du *shell* : l'interpréteur vérifie d'abord que le programme est syntaxiquement correct, puis l'exécute.
- *Python* est un langage objet interprété précompilé. Une première passe du programme *python* sur le script vérifie la grammaire du code et crée un fichier *.pyc*. Cette forme précompilée est sauvegardée lors de l'exécution du script. Dans un deuxième temps ce code précompilé est exécuté par la composante de *python* qui joue le rôle de machine virtuelle.

16.4 Installer Python

16.4.1 Sous Windows

Sur le site web officiel de *Python* : `http://www.python.org/`, vous trouverez dans la section *Download* des logiciels d'installation automatique pour les différentes versions de *Python*. Vous pouvez en confiance choisir la dernière version mais certains modules ne sont alors pas forcément disponibles.

Par exemple, au 20 *aot* 2017, il s'agissait de la version 3.6.2.

Copiez ce fichier dans un répertoire temporaire de votre ordinateur et exécutez-le. *Python3* s'installe par défaut dans un répertoire nommé Python** (où ** indique les deux premiers chiffres du n^o de version), et des icônes de lancement seront mises en place automatiquement.

Si vous voulez bénéficier des ressources de bibliothèques tierces qui ne sont pas encore disponibles pour *Python3* (*PIL*, *urllib*), vous pouvez sans crainte installer aussi la dernière version de *Python2*, en téléchargeant le fichier *Python* 2.7.2 *Windows Installer* (ou son équivalent 64 bits).

Les deux versions s'installeront dans des répertoires différents et ne se gêneront en aucune manière.

Lorsque l'installation est terminée, vous pouvez effacer le contenu du répertoire temporaire.

16.4.2 Sous Linux

Vous avez probablement installé votre système *Linux* à l'aide d'une distribution telle que *Ubuntu*, *SuSE*, *RedHat*, *dots*

Installez simplement les paquetages *Python* à l'aide de la logithèque.

Si vous installez *Python2* et *Python3*, il vous faudra aussi parfois installer les 2 versions correspondantes des modules.

Certains modules devront être installés à la main.

Vous pourrez également visiter le site : `http://www.python.org/getit/`.

16.4.3 Sous Mac OS

Sur le site officiel de *Python*, vous trouverez des paquetages installeurs pour *Mac OS* similaires à ceux qui sont proposés pour *Windows*.

16.5 Développer en Python

À partir du moment où *Python* est installé sur un ordinateur (*Linux*, *Windows* ou *MacOS*), taper "*python3*" dans un terminal donne accès à l'interpréteur interactif de*Python*. Il ne dispose toutefois pas des bibliothèques mathématiques, il faut les importer (*cf.* liste de `import` de la page 302). On peut alors utiliser *Python* comme une calculatrice évoluée, en ligne de commande.

On peut également lancer un programme *Python* écrit dans un fichier texte *exemple.py* par : `python exemple.py`

On notera qu'ensuite le répertoire contient aussi un fichier binaire *exemple.pyc*. On pourrait même mesurer qu'une seconde exécution se fait plus rapidement : l'étape de précompilation n'a pas à être faite.

Si un programme nécessite des paramètres, on pourra les lui passer par la ligne de commande (ici avec le fichier *eratosthene.py* dont le listing est ci-dessous et le paramètre 200) :

`python eratosthene.py 200`.

eratosthene.py

```
1  #! /usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  N = int(sys.argv[1]) #nécéssite une commande du type python3
                                        eratosthene.py 200
7  premiers = [x for x in range(0,N)]
8  for n in range(2,int(N**0.5)):
9      for i in range(2*n, N, n):
10         premiers[i]=0
11 premiers[1]=0
12 for i in premiers:
13     if premiers[i] != 0:
14         print (i,end=' ')
```

16.5.1 GNU emacs

GNU emacs est un éditeur de code informatique universel, adapté à tous les langages. Son interface est essentiellement sous forme de texte, et il s'appuie fortement sur des raccourcis clavier qui lui sont spécifiques. Il est disponible sur toutes les plate-formes, mais mieux intégré au monde *Linux*. Avant l'arrivée d'*Eclipse*, il s'agissait de l'éditeur le plus utilisé dans la plupart des écoles.

16.5.2 ActivePython

ActivePython est disponible sous *Windows*.

- Pour l'installer, lancer le logiciel d'installation *ActivePython-3.2.2.3-win32-x86* ou une version plus récente. Il va installer *IDLE* dans le répertoire *C :/Python32*.
- On pourra ensuite installer *matplotlib* en lançant *matplotlib-1.2.0.win32-py3.2* (cette bibliothèque sera installée dans *C :/Python32*)
- L'installation de la bibliothèque *Numpy* dans ce même répertoire se fera en lançant *numpy-1.7.0-win32-superpack-python3.2*.
- L'installation de la bibliothèque *Scipy* encore dans ce répertoire se fera en lançant *scipy-0.11.0-win32-superpack-python3.2*.
- On pourra installer la bibliothèque *Sympy* en copiant tout simplement le répertoire sous *C :/Python32/Lib/site-packages*.

16.5.3 Eclipse/PyDev

16.5.3.1 Description

L'IDE *Eclipse* que l'on peut télécharger ici : http://www.eclipse.org/, permet d'écrire du code dans de nombreux langages. Il fournit une coloration syntaxique, de l'indentation automatique, de la complétion automatique, et un contrôle de l'exécution pour le débogage. Il gère les ensembles de fichier sources sous la forme de *projet* et est adapté aux développements de grands logiciels en équipe. Initialement écrit pour *Java* (en *Java*), le module *PyDev* téléchargeable sur : http://pydev.org/ permet de l'utiliser pour *Python*. Il s'agit de l'IDE le plus utilisé ces dernières années, et de nombreuses écoles l'ont adopté. Il est disponible sur toutes les plate-formes.

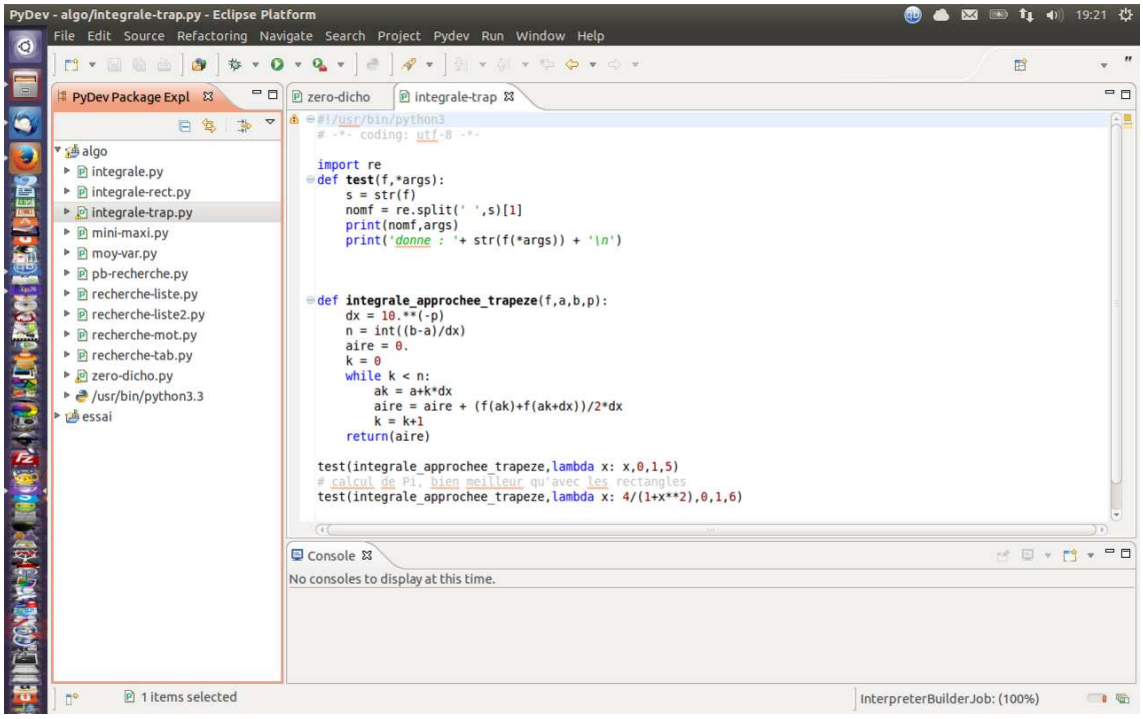


FIGURE 16.1 – Fenêtre Eclipse

16.5.3.2 Installation

Installation de *Eclipse* + *PyDev* :

- Sous *Linux*
  Le plus simple, une fois que *Python* est installé, est de passer en mode Terminal (CTRL ALT T) et de taper *sudo apt-get install eclipse*. Pour installer *PyDev*, il suffit d'ouvrir *Eclipse*, puis de sélectionner *Help* > *Install New Software...*. Dans la fenêtre qui apparaît, cliquez sur *Add...* pour ajouter une nouvelle source.
  Entrez alors les informations suivantes :
  - Name : *PyDev*
  - Location : *http ://pydev.org/updates/*

  Eclipse parse alors les plugins disponibles à cette adresse. Sélectionnez *PyDev for Eclipse* et suivez les instructions pour l'installer.

  Il faut d'abord faire reconnaitre l'interpréteur *Python* avec *Windows* > *Preference* > *Interpreter - Python*. Choisir *new* > *browse*. Faire *Apply* et quitter.
  Sélectionnez (par exemple) */usr/bin/python3.3*.
  Il ne reste plus qu'à ajouter quelques points de configuration, en particulier concernant l'encodage souhaité grâce à *Windows* > *Preference* > *General* > *workspace*.
  On définit ici l'encodage : *UTF8* pour gérer les accents.
  Redémarrez *Eclipse* comme demandé.

- Sous *Windows*
  Pour installer *Eclipse*, il suffit de décompresser l'archive zip ou tar.gz téléchargée. Cela crée un dossier, nommé *Eclipse*, qu'il est conseillé de placer aussi haut que possible dans la hiérarchie de fichiers du système. Dans la suite de cette note, on supposera qu'il est placé dans un dossier nommé *C :\ eclipse*.
  L'idéal est ensuite de placer un raccourci sur le bureau.
  On lance *Eclipse* en lançant *c :\ eclipse\ eclipse.exe*.
  Au cours du lancement, *Eclipse* demande un "workspace" (espace de travail sur disque). Pour travailler avec *Python*, vous pouvez créer un répertoire personnel qui contiendra tous les projets *Python*.

  Pour intégrer *Pydev* après avoir lancé *Eclipse*, faire *help* > *install new software* puis en ligne de saisie *work with*. Ajouter `http://pydev.org/updates` et cliquez sur le bouton *add*.
  Suivez les instructions suivantes.
  Pydev est alors installé, mais cela ne suffit pas : Il faut d'abord faire reconnaitre l'interpréteur *Python* avec *Windows* > *Preference* puis *Pydev* > *Interpreter - Python*. Choisir *new* > *browse*. Faire *Apply* et quitter.
  Sélectionnez (par exemple) *c :\ Python32\ python3.2.exe*.
  Il ne reste plus qu'à ajouter quelques points de configuration, en particulier concernant l'encodage souhaité grâce à *Windows* > *Preference* > *General* > *workspace*.
  On définit ici l'encodage : *UTF8* pour gérer les accents.
  Redémarrez *Eclipse* comme demandé.

16.5.3.3 Utilisation de *Eclipse*

- Pour commencer un projet, cliquez sur l'onglet *File* > *New* > *Project* > *PyDev Project*, puis par exemple *Beutier*. N'oubliez pas de choisir la version *python3* pour la *Grammar Version*

  Placez-vous sur le projet (ici *Beutier*) puis créez un fichier en faisant "clic droit", *New*, puis un nom de fichier, par exemple *toto.py*.
  En compilant via la commande *Run* ou *Run as* > *Python*, le fichier *toto* est sauvegardé dans l'espace de travail ("workspace") défini au début.

- Pour ouvrir un fichier existant mais qui n'a pas été créé par *Eclipse*, vous pouvez ouvrir le répertoire dans lequel se trouve le fichier, puis le faire glisser dans le projet (ici *Beutier*).
  Ce premier fichier ne sera alors pas sauvegardé : une copie sera sauvegardée dans le "workspace" (espace de travail) défini au début.

16.5.4 geany

16.5.4.1 Description

Geany est un éditeur de texte utilisant *GTK2* avec des fonctions basiques d'environnement de développement intégré (EDI). Il a été développé pour fournir un EDI rapide et simple qui n'a que peu de dépendances. *Geany* est disponible sur toutes les plate-formes et intègre des fonctionnalités assez sympathiques :

- coloration syntaxique,
- code source « pliable »,
- auto-complétion sur les structures souvent utilisées comme : if, for et while,
- auto-complétion des balises XML et HTML,
- trucs et astuces,
- copie de la ligne ou de la sélection courantes en une seule action,
- support de nombreux types de fichiers comme *C*, *C++*, *CSS*, *Java*, *LaTeX*, *PHP*, *Python*, *Perl*, *Pascal*, *Ruby*, *SQL*, ...
- listes des variables et fonctions utilisées,
- émulateur de terminal pour l'exécution du programme sans quitter l'éditeur et/ou pour l'entrée de commandes.

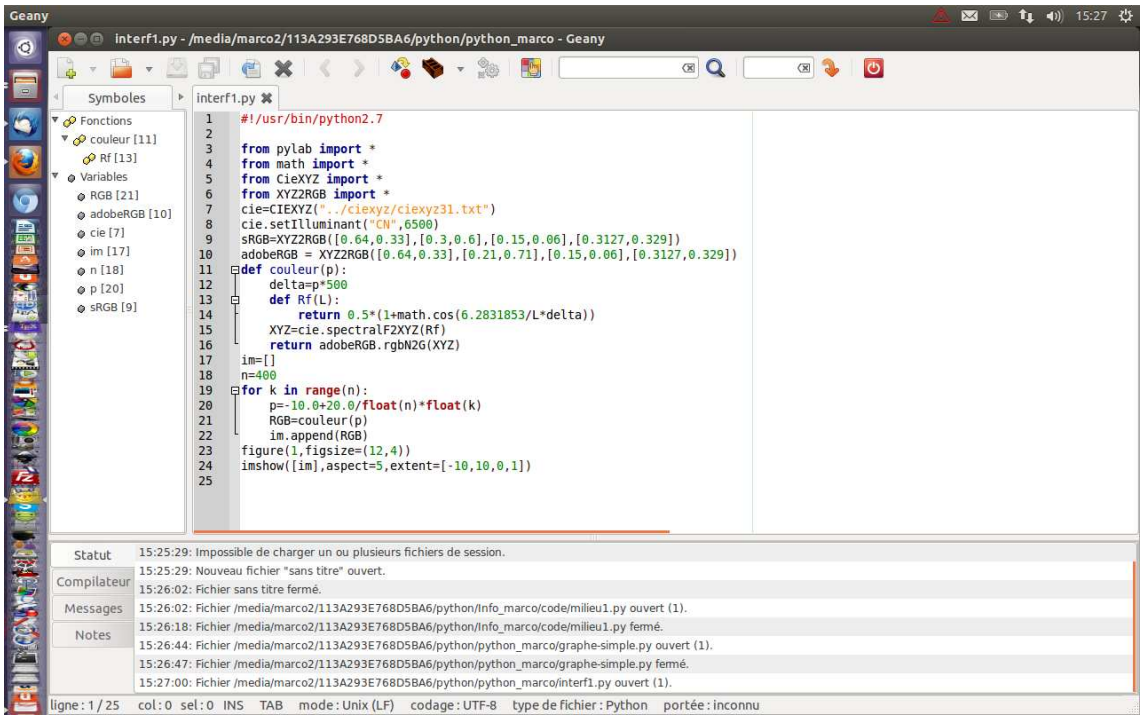


FIGURE 16.2 – Fenêtre geany

16.5.4.2 Installation

L'installation sous *Linux* se fait via les dépôts officiels, directement avec la logithèque d'*Ubuntu*.
Pour l'installer sous *Windows*, il faut télécharger le logiciel *geany-1.23.1_setup.exe* à l'adresse

http://www.geany.org/Download/Releases

et le lancer.

Si vous avez un message de ce type sous *Windows* :
" 'python' n'est pas reconnu en tant que commande interne ou externe, un programme exécutable ou un fichier de commande.
Appuyez sur une touche pour continuer . . . ",
il faudra mettre *Python* dans le path. Par exemple, sous *Windows XP* :

- Cliquez droit sur "Poste de Travail"
- Choisissez "Propriétés"
- Allez dans l'onglet "Avancé"
- Puis cliquez sur "Variables d'environnement"
- Dans le cadre "Variables système", cliquez sur "Path" puis sur "modifier"
- Ajoutez à la fin de la ligne "Valeur de la variable" le séparateur ';' suivi du chemin où vous avez installé Python (par exemple : *C :\ Program Files \ Python*)

Sous *Windows 7*, le même type de manipulation peut se faire ; il suffit de trouver : "Variables d'environnement".

16.5.5 Spyder

C'est l'éditeur que nous utiliserons si possible !

16.5.5.1 Sous *Windows*

Spyder est téléchargeable ici :

https://sourceforge.net/projects/winpython/files/WinPython_3.6/3.6.7.0/

ou bien ici :

https://bitbucket.org/spyder-ide/spyderlib/downloads/

Très orienté calcul mathématique (précharge par défaut les librairies *SciPy*, *NumPy*, *Matplotlib*), *WinPython* existe en version 32 bits et 64 bits : choisissez celle qui vous convient. L'avantage de ce package, c'est qu'il n'est pas nécessaire d'installer *Python* par ailleurs : cette solution est déjà très complète.

Lors de l'installation, il est préférable d'installer *WinPython* à la racine du disque dur principal. Ensuite, il faut lancer l'application *Spyder* : il peut être judicieux de placer un raccourci sur le bureau.

Il est également possible d'installer *WinPython* sur une clé USB ou un disque dur externe mais son exécution peut être très lente.

16.5.5.2 Sous *Ubuntu*

Une méthode pour installer *Spyder* sous *ubuntu*, c'est de taper, dans un terminal :

```
sudo apt install python3
sudo apt install python3-pip
sudo apt install ipython3
sudo apt install ipython3-qtconsole
sudo pip3 install -U numpy
sudo apt-get build-dep python-scipy
sudo pip3 install -U scipy
sudo pip3 install -U sympy
sudo pip3 install -U matplotlib

sudo pip3 install pep8
sudo pip3 install pyflakes
sudo pip3 install sphinx
sudo pip3 install rope_py3k

cd ~
mkdir 00Software
cd 00Software
wget https://bitbucket.org/spyder-ide/spyderlib/downloads/spyder-2.3.9.zip
unzip spyder-2.3.9.zip
cd spyder-2.3.9
python3 setup.py build
sudo python3 setup.py install
sudo apt-get install python3-dev
```

Pour installer une nouvelle bibliothèque, par exemple *CoolProp* pour les diagrammes thermodynamiques, vous pourrez faire ensuite :

```
sudo pip3 install Cython
sudo pip3 install Coolprop
```

Sur une même machine peuvent cohabiter *Spyder 3* et *Spyder 2*. Cela peut être intéressant pour certains programmes qui ne fonctionnent pas sous *Python 3*.

16.6 Obtenir de l'aide

Il faut savoir tout d'abord que de l'aide sur les différentes fonctions de *python* pourra être obtenue grâce à la commande *help()*. En effet, vous pouvez par exemple taper :

```
help(exit)
```

Pour sortir de cette aide, il vous suffira de taper "q".

Vous pourrez également se référer à des sites aussi divers que variés, par exemple :

- http://fr.openclassrooms.com/informatique/python/cours
- https://wiki.python.org/moin/FrenchLanguage
- http://docs.python.org/3/
- http://www.olivierberger.org/python/doc/tut/tut.html
- http://www.jchr.be/python/manuel.htm
- http://python.jpvweb.com/mesrecettespython/doku.php
- http://python.developpez.com/faq/
- http://python.developpez.com/cours/TutoSwinnen/?page=page_1
- http://inforef.be/swi/python.htm (la même chose que précédemment, mais en pdf)

La liste est bien sûr sans fin mais les principaux sont là.
Pour les bibliothèques, ce sera vu en cours d'apprentissage.

16.7 Modules installés

Voici une méthode pour connaître les modules installés sur votre machine ainsi que leur version :

16.7.1 Sous *Ubuntu*

Il faut d'abord installer *pip*. Dans une console, tapez :

```
sudo apt install python3-pip
```

Ainsi, vous obtenez par exemple :

```
Brlapi==0.6.0
CoolProp==4.2.5
Cython==0.20.2
Jinja2==2.7.3
MarkupSafe==0.23
Pillow==2.0.0
Pygments==1.6
Sphinx==1.2.2
apparmor==2.8.0
apturl==0.5.2ubuntu2
chardet==2.0.1
command-not-found==0.3
decorator==3.4.0
defer==1.0.6
dirspec==13.10
distribute==0.6.37
docutils==0.12
feedparser==5.1.2
friends==0.1
httplib2==0.8
ipython==0.13.2
language-selector==0.1
louis==2.5.3
lxml==3.2.0
matplotlib==1.2.1
mpmath==0.17
nose==1.3.0
numpy==1.8.1
oauthlib==0.5.1
onboard==0.99.0
oneconf==0.3.5
pep8==1.5.7
piston-mini-client==0.7.5
pycrypto==2.6
pyflakes==0.8.1
pygobject==3.10.0
pyparsing==2.0.1
pysmbc==1.0.13
python-apt==0.8.9.1ubuntu1
python-dateutil==2.0
python-debian==0.1.21-nmu2ubuntu1
python-pipeline==0.1.3-py3k
pytz==2012c
pyxdg==0.25
pyzmq==13.1.0
scipy==0.14.0
simplegeneric==0.8.1
simpy==3.0.5
six==1.3.0
software-center-aptd-plugins==0.0.0
spyder==2.3.0
sympy==0.7.5
thin-client-config-agent==0.7
ubuntu-drivers-common==0.0.0
ufw==0.33-0ubuntu4
unattended-upgrades==0.1
unity-scope-audacious==0.1
unity-scope-calculator==0.1
unity-scope-chromiumbookmarks==0.1
unity-scope-clementine==0.1
unity-scope-colourlovers==0.1
unity-scope-devhelp==0.1
unity-scope-firefoxbookmarks==0.1
unity-scope-gdrive==0.7
unity-scope-gmusicbrowser==0.1
unity-scope-gourmet==0.1
unity-scope-guayadeque==0.1
unity-scope-manpages==0.1
unity-scope-musique==0.1
unity-scope-openclipart==0.1
unity-scope-texdoc==0.1
unity-scope-tomboy==0.1
unity-scope-virtualbox==0.1
unity-scope-yelp==0.1
unity-scope-zotero==0.1
usb-creator==0.2.23
xdiagnose==3.6.1
xkit==0.0.0
```

Vous pouvez également utiliser, en console *IPython*, la commande *help('modules')*.Vous obtenez ainsi par exemple :

```
>>> help('modules')

Please wait a moment while I gather a list of all available modules...

Started our command server
/usr/lib/python3/dist-packages/UnityTweakTool/__init__.py:40: PyGIWarning: Gtk was
    imported without specifying a version first. Use gi.require_version('Gtk', '3.0'
    ) before import to ensure that the right version gets loaded.
  from gi.repository import Gtk
/usr/lib/python3/dist-packages/matplotlib/cbook.py:137: MatplotlibDeprecationWarning
    : The matplotlib.delaunay module was deprecated in version 1.4. Use matplotlib.
    tri.Triangulation instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/usr/lib/python3/dist-packages/pyatspi/__init__.py:17: PyGIWarning: Atspi was
    imported without specifying a version first. Use gi.require_version('Atspi', '2.
    0') before import to ensure that the right version gets loaded.
  from gi.repository import Atspi
/usr/lib/python3.5/pkgutil.py:86: FutureWarning: The pandas.rpy module is deprecated
     and will be removed in a future version. We refer to external packages like
    rpy2.
See here for a guide on how to port your code to rpy2: http://pandas.pydata.org/
    pandas-docs/stable/r_interface.html
  __import__(name)
/usr/lib/python3/dist-packages/usbcreator/backends/udisks/backend.py:4: PyGIWarning:
     UDisks was imported without specifying a version first. Use gi.require_version(
    'UDisks', '2.0') before import to ensure that the right version gets loaded.
  from gi.repository import Gio, GLib, UDisks
/usr/lib/python3/dist-packages/usbcreator/frontends/gtk/unitysupport.py:27:
    PyGIWarning: Unity was imported without specifying a version first. Use gi.
    require_version('Unity', '7.0') before import to ensure that the right version
    gets loaded.
  from gi.repository import Unity
AptUrl              audioop             jdcal               sanstitre10
CDROM               autoreload          jedi                sanstitre2
CommandNotFound     axi                 jinja2              sanstitre3
CoolProp            babel               json                sanstitre4
Crypto              base64              jwt                 sanstitre5
DLFCN               bdb                 keyword             sanstitre6
DistUpgrade         binascii            language_support_pkgs sanstitre7
IN                  binhex              lib2to3             sanstitre8
IPython             bisect              linecache           sanstitre9
LanguageSelector    blinker             locale              scanext
NvidiaDetector      brlapi              logging             sched
Onboard             bs4                 louis               scipy
OpenSSL             builtins            lsb_release         seawater
PIL                 bz2                 lxml                select
PyPDF2              cProfile            lzma                selectors
PyQt4               cairo               macpath             sessioninstaller
PyQt5               calendar            macurl2path         setuptools
Quirks              cesar               mailbox             shelve
TYPES               cgi                 mailcap             shlex
UbuntuDrivers       cgitb               mako                shutil
UbuntuSystemService chardet             markdown            signal
UnityTweakTool      checkbox_ng         markupsafe          simplegeneric
UpdateManager       checkbox_support    marshal             sip
Xlib                chunk               math                sipconfig
__future__          cmath               matplotlib          sipconfig_d5
_ast                cmd                 mimetypes           sipconfig_nd5
_bisect             code                mmap                sipdistutils
_bootlocale         codecs              modulefinder        site
_bz2                codeop              mpl_toolkits        sitecustomize
_cffi_backend       collections         multiprocessing     six
_codecs             colorsys            ncepgrib2           smbc
_codecs_cn          compileall          netCDF4             smtpd
_codecs_hk          concurrent          netcdftime          smtplib
_codecs_iso2022     configparser        netrc               sndhdr
_codecs_jp          contextlib          nis                 socket
_codecs_kr          copy                nntplib             socketserver
_codecs_tw          copyreg             nose
    softwarecenter_aptd_plugins
```

```
_collections          crypt                 ntpath                softwareproperties
_collections_abc      cryptography          nturl2path            speechd
_compat_pickle        csv                   numbers               speechd_config
_compression          ctypes                numexpr               sphinx
_crypt                cups                  numpy                 sphinx_rtd_theme
_csv                  cupsext               oauthlib              spwd
_ctypes               cupshelpers           occur                 spyderlib
_ctypes_test          curl                  oceans                spyderplugins
_curses               curses                octavemagic           sqlite3
_curses_panel         cx_Freeze             oneconf               sre_compile
_datetime             cycler                opcode                sre_constants
_dbm                  cythonmagic           openpyxl              sre_parse
_dbus_bindings        datetime              operator              ssl
_dbus_glib_bindings   dateutil              optparse              stat
_decimal              dbm                   orca                  statistics
_dummy_thread         dbus                  os                    storemagic
_elementtree          deb822                ossaudiodev           string
_functools            debconf               padme                 stringprep
_gdbm                 debian                pandas                struct
_hashlib              debian_bundle         parallelmagic         subprocess
_heapq                decimal               parser                sunau
_imp                  decorator             pathlib               symbol
_io                   defer                 pcardext              sympyprinting
_json                 difflib               pdb                   symtable
_locale               dis                   pep8                  sys
_lsprof               distutils             pexpect               sysconfig
_lzma                 doctest               pickle                syslog
_markupbase           docutils              pickletools           systemd
_md5                  dummy_threading       piexif                tables
_multibytecodec       easy_install          pip                   tabnanny
_multiprocessing      email                 pipes                 tarfile
_opcode               encodings             piston_mini_client    telnetlib
_operator             enum                  pkg_resources         tempfile
_osx_support          errno                 pkgutil               termios
_pickle               espeak                plainbox              test
_portaudio            ess-csv-web           platform              textwrap
_posixsubprocess      et_xmlfile            plistlib              this
_pydecimal            etoiles               poplib                threading
_pyio                 exifread              posix                 time
_random               faulthandler          posixpath             timeit
_sha1                 fcntl                 pprint                tkinter
_sha256               feedparser            problem_report        token
_sha512               feedparser_sgmllib3   profile               tokenize
_signal               filecmp               proie                 trace
_sitebuiltins         fileinput             pstats                traceback
_smbc                 flask                 psutil                tracemalloc
_socket               flask_flatpages       pty                   try
_sqlite3              fnmatch               ptyprocess            tty
_sre                  formatter             pwd                   turtle
_ssl                  fpectl                py_compile            types
_stat                 fractions             pyasn1                typing
_string               ftplib                pyatspi               ufw
_strptime             functools             pyaudio               unicode
_struct               g2clib                pyclbr                unicodedata
_symtable             gc                    pycurl                unittest
_sysconfigdata        genericpath           pydblite              uno
_sysconfigdata_dm     getopt                pydoc                 unohelper
_sysconfigdata_m      getpass               pydoc_data            urllib
_testbuffer           gettext               pyexpat               urllib3
_testcapi             gi                    pyflakes              usbcreator
_testimportmultiple   glob                  pygame                uu
_testmultiphase       gpxpy                 pygments              uuid
_thread               grp                   pygrib                venv
_threading_local      gsw                   pygtkcompat           warnings
_tkinter              gtts                  pyinotify             wave
_tracemalloc          gtts_token            pykeyboard            weakref
_warnings             guacamole             pylab                 webbrowser
_weakref              gzip                  pymouse               werkzeug
_weakrefset           hashlib               pyparsing             wheel
_yaml                 heapq                 pyproj                wsgiref
abc                   hmac                  pytz                  xapian
aifc                  hpmudext              pyzo                  xdg
alabaster             html                  queue                 xdiagnose
antigravity           html5lib              quopri                xdrlib
apport                http                  random                xkit
apport_python_hook    httplib2              re                    xlrd
apt                   idlelib               readline              xlsxwriter
```

```
apt_inst            idna                redtoreg            xml
apt_pkg             imaplib             reportlab           xmlrpc
aptdaemon           imghdr              reprlib             xxlimited
aptsources          imp                 requests            xxsubtype
argparse            importlib           resource            yaml
array               inspect             rlcompleter         zipapp
ast                 io                  rmagic              zipfile
asynchat            ipaddress           roman               zipimport
asyncio             itertools           runpy               zlib
asyncore            itsdangerous        sanstitre0          zmq
atexit              janitor             sanstitre1

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose name or summary contain the string "spam".
```

16.7.2 Sous *Windows*

En console, tapez *help('modules')*. Vous obtenez par exemple :

```
>>> help('modules')

Please wait a moment while I gather a list of all available modules...

Expected Tk Togl installation in C:\WinPython-64bit-3.3.3.3\python-3.3.3.amd64\lib\
    site-packages\OpenGL\Tk\togl-win32-64
C:\WinPython-64bit-3.3.3.3\python-3.3.3.amd64\lib\site-packages\sympy\statistics\
    __init__.py:11: SymPyDeprecationWarning:

sympy.statistics has been deprecated since SymPy 0.7.2. Use
sympy.stats instead. See
http://code.google.com/p/sympy/issues/detail?id=3386 for more info.

  deprecated_since_version="0.7.2",
Cython              cythonmagic         perfmon             sspi
IPython             datetime            pickle              sspicon
OpenGL              dateutil            pickleshare         sstruct
PIL                 dbi                 pickletools         stat
Polygon             dbm                 pilconvert          statsmodels
PyPDF2              dde                 pildriver           storemagic
PyQt4               decimal             pilfile             string
PySide              difflib             pilfont             stringprep
__future__          dis                 pilprint            struct
_ast                distutils           pip                 subprocess
_bisect             doctest             pip-script          sunau
_bz2                docutils            pip3-script         symbol
_codecs             dummy_threading     pipes               symilar-script
_codecs_cn          easy_install        pkg_resources       sympy
_codecs_hk          easy_install-script pkgutil             sympyprinting
_codecs_iso2022     email               platform            symtable
_codecs_jp          encodings           plistlib            sys
_codecs_kr          epylint-script      poplib              sysconfig
_codecs_tw          errno               posixpath           tables
_collections        f2py                pprint              tabnanny
_compat_pickle      faulthandler        profile             tarfile
_csv                filecmp             pstats              telnetlib
_ctypes             fileinput           psutil              tempfile
_ctypes_test        fnmatch             pty                 test
_datetime           fontTools           py2gi               tests
_decimal            formatter           py2mechanize        textwrap
_dummy_thread       formlayout          py2qt4              this
_elementtree        fractions           py2stdlib           threading
_functools          ftplib              py_compile          time
_hashlib            functools           pyclbr              timeit
_heapq              gc                  pycolor3-script     timer
_imp                genericpath         pydoc               tkinter
_io                 getopt              pydoc_data          token
_json               getpass             pyexpat             tokenize
_locale             gettext             pyflakes            tornado
_lsprof             glob                pygmentize-script   trace
```

```
_lzma                guidata              pygments             traceback
_markerlib           guiqwt               pyhdf                ttfquery
_markupbase          gzip                 pylab                tty
_md5                 h5py                 pylint               turtle
_msi                 hashlib              pylint-gui-script    turtledemo
_multibytecodec      heapq                pylint-script        types
_multiprocessing     hmac                 pyparsing            unicodedata
_osx_support         html                 pyreadline           unittest
_pickle              http                 pyreverse-script     urllib
_psutil_mswindows    idlelib              pyside-uic-script    uu
_pyio                imaplib              pyside_postinstall   uuid
_random              imghdr               pysideuic            venv
_sha1                imp                  pythoncom            vidle
_sha256              importlib            pytz                 vis
_sha512              inspect              pywin                visual
_socket              io                   pywin32_postinstall  warnings
_sqlite3             ipaddress            pywin32_testall      wave
_sre                 ipcluster3-script    pywin32_testutil     weakref
_ssl                 ipcontroller3-script pywintypes            webbrowser
_string              ipengine3-script     pywt                 win2kras
_strptime            iplogger3-script     pyximport            win32api
_struct              iptest3-script       queue                win32clipboard
_symtable            ipython3-script      quopri               win32com
_testbuffer          ipython_win_post_install random                win32con
_testcapi            irunner3-script      rasutil              win32console
_thread              isapi                re                   win32cred
_threading_local     itertools            readline             win32crypt
_tkinter             jinja2               regcheck             win32cryptcon
_warnings            json                 regutil              win32event
_weakref             keyword              reprlib              win32evtlog
_weakrefset          lib2to3              rlcompleter          win32evtlogutil
_win32sysloader      linecache            rmagic               win32file
_winapi              locale               rope                 win32gui
_winxptheme          lockfile             rst2html             win32gui_struct
abc                  logging              rst2latex            win32help
adodbapi             logilab              rst2man              win32inet
afxres               lzma                 rst2odt              win32inetcon
aifc                 macpath              rst2odt_prepstyles   win32job
antigravity          macurl2path          rst2pseudoxml        win32lz
argparse             mahotas              rst2s5               win32net
array                mailbox              rst2xetex            win32netcon
ast                  mailcap              rst2xml              win32pdh
astroid              markupsafe           rstpep2html          win32pdhquery
asynchat             marshal              runpy                win32pdhutil
asyncore             math                 runxlrd              win32pipe
atexit               matplotlib           sched                win32print
audioop              matplotlibwidget     scipy                win32process
autoreload           mimetypes            select               win32profile
base64               miniterm             serial               win32ras
bdb                  mmap                 servicemanager       win32rcparser
binascii             mmapfile             setuptools           win32security
binhex               mmsystem             shelve               win32service
bisect               modulefinder         shlex                win32serviceutil
builtins             msilib               shutil               win32timezone
bz2                  msvcrt               signal               win32trace
cProfile             multiprocessing      simplejson           win32traceutil
calendar             netbios              sip                  win32transaction
cgi                  netrc                sipconfig            win32ts
cgitb                networkx             sipdistutils         win32ui
chunk                nntplib              site                 win32uiole
cmath                nose                 six                  win32verstamp
cmd                  nosetests-script     skimage              win32wnet
code                 nt                   skivi-script         winerror
codecs               ntpath               sklearn              winioctlcon
codeop               ntsecuritycon        smtpd                winnt
collections          nturl2path           smtplib              winperf
colorama             numbers              sndhdr               winpython
colorsys             numexpr              socket               winreg
commctrl             numpy                socketserver         winsound
```

```
compileall          octavemagic         sphinx              winxpgui
concurrent          odbc                sphinx-apidoc-script winxptheme
configparser        opcode              sphinx-autogen-script wsgiref
contextlib          operator            sphinx-build-script xdrlib
copy                optparse            sphinx-quickstart-script xlrd
copyreg             oral ccs pendule    spyder_win_post_install xml
crypt               os                  spyderlib           xmlWriter
csv                 os2emxpath          spyderplugins       xmlrpc
ctypes              pandas              sqlalchemy          xxsubtype
curses              parallelmagic       sqlite3             zipfile
cx_Freeze           parser              sre_compile         zipimport
cygdb-script        path                sre_constants       zlib
cython              patsy               sre_parse           zmq
cython-script       pdb                 ssl

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose descriptions contain the word "spam".
```

16.8 Ajouter des modules

Quelques modules intéressants à ajouter :

- `PyPDF2` pour assembler des fichiers *pdf*,
- `CooProp` pour représenter des diagrammes de fluides,
- `imaplib`, `imaplib` pour les mails,
- `ftplib` pour travailler en *ftp*,
- `bs4` pour parser le *html*,
- ...

16.8.1 Sous Linux

Par exemple, pour installer `NomDuModule`, on peut utiliser les instructions :

```
sudo apt-get install python3-NomDuModule
```

16.8.1.1 Installation de pip

Une autre option consiste à utiliser `pip` :

```
sudo apt install python3-setuptools
sudo easy_install3 pip
```

ou alors :

```
sudo apt install python3-pip
```

puis ensuite :

```
sudo apt install python3-setuptools
sudo easy_install3 pip
```

16.8.1.2 Utilisation de pip

```
sudo pip3 install NomDuModule
```

16.8.2 Sous Windows

16.8.2.1 Installation de pip

Pour installer `NomDuModule`,on peut utiliser `pip`.
Pour cela, atteindra la console grâce à la touche `WINDOWS` puis saisir `cmd`. Ensuite, atteindre le dossier `scripts`, ici en faisant :

```
cd C:\WinPython-64bit-3.3.3.3\python-3.3.3.amd64\Scripts
```

Ce chemin est bien entendu à adapter en fonction de votre configuration...

16.8.2.2 Utilisation de pip

Pour installer le module en question, il suffit alors de taper

```
pip3.3 install NomDuModule
```

Évidemment, le nom de fichier `pip3.3` peut varier également.

16.9 Conclusion

Comme nous l'avons vu, *Python* peut fonctionner en console, ce qui est très simple sous *Linux* et bien adapté aux calculs mais dès lors que l'on souhaite modifier et / ou conserver ses programmes, il est plus judicieux de travailler avec un éditeur. Nous utiliserons donc la console ou l'éditeur suivant le travail à effectuer. L'avantage de *Spyder* est de présenter les 2 dans la même fenêtre.

17

Manipulation de fichiers externes

17.1 Travailler avec des fichiers

L'utilisation d'un fichier ressemble beaucoup à celle d'un livre. Pour utiliser un livre, vous devez d'abord le trouver (à l'aide de son titre), puis l'ouvrir. Lorsque vous avez fini de l'utiliser, vous le refermez. Tant qu'il est ouvert, vous pouvez y lire des informations diverses, et vous pouvez aussi y écrire des annotations, mais généralement vous ne faites pas les deux à la fois. Dans tous les cas, vous pouvez vous situer à l'intérieur du livre, notamment en vous aidant des numéros de pages. Vous lisez la plupart des livres en suivant l'ordre normal des pages, mais vous pouvez aussi décider de consulter n'importe quel paragraphe dans le désordre.

Tout ce que nous venons de dire des livres s'applique également aux fichiers informatiques. Un fichier se compose de données enregistrées sur votre disque dur, une clef USB, un CD, ...
Vous y accédez grâce à son nom (lequel peut inclure aussi un nom de répertoire). En première approximation, vous pouvez considérer le contenu d'un fichier comme une suite de caractères, ce qui signifie que vous pouvez traiter ce contenu, ou une partie quelconque de celui-ci, à l'aide des fonctions servant à traiter les chaînes de caractères.

17.2 Écriture séquentielle dans un fichier

Sous *Python*, l'accès aux fichiers est assuré par l'intermédiaire d'un objet-interface particulier, que l'on appelle objet-fichier. On crée cet objet à l'aide de la fonction intégrée `open()`. Celle-ci renvoie un objet doté de méthodes spécifiques, qui vous permettront de lire et écrire dans le fichier.

On crée un objet fichier Python avec la fonction `open()` : Notez bien que si le fichier n'existe pas encore, il sera créé automatiquement. Par contre, si le nom utilisé concerne un fichier préexistant qui contient déjà des données, les caractères que vous y enregistrerez viendront s'ajouter à la suite de ceux qui s'y trouvent déjà.

```
>>> MonFichier = open("NomDuFichier",'r')
```

Le mode peut être 'r' (read, en lecture seule), 'w' (en écriture seule), 'a' pour rajouter à la fin du fichier, ...
D'autres modes (mixte lecture/écriture, etc...) existent : ils sont indiqués ci-après. La lecture des fichiers texte (c'est-à-dire non binaires) se fait avec les fonctions `read()` pour lire le fichier, `readline()` pour la ligne courante et `readlines()` pour les différentes lignes.

Mode	Description du mode d'ouverture
`r`	Ouvre un fichier texte en mode lecture : le fichier doit exister.
`w`	Ouvre un fichier texte en mode écriture : si le fichier existe, son contenu est écrasé.
`a`	Ouvre un fichier texte en mode ajout pour écrire à la fin du fichier. La fonction *fopen()* crée le fichier s'il n'existe pas.
`r+`	Ouvre un fichier texte en mode lecture et écriture : le fichier doit exister.
`w+`	Ouvre un fichier texte en mode lecture et écriture : si le fichier existe, son contenu est écrasé.
`a+`	Ouvre un fichier texte en mode ajout pour lecture ou mise à jour à la fin du fichier. La fonction `open()` crée le fichier s'il n'existe pas.
`rb`	Ouvre un fichier binaire en mode lecture : le fichier doit exister.
`wb`	Ouvre un fichier binaire vide en mode écriture. Si le fichier existe, son contenu est écrasé.
`ab`	Ouvre un fichier binaire en mode ajout pour écrire à la fin du fichier. La fonction `open()` crée le fichier s'il n'existe pas.
`r+b` ou `rb+`	Ouvre un fichier binaire en mode lecture et écriture : le fichier doit exister.
`w+b` ou `wb+`	Ouvre un fichier binaire vide en mode lecture : si le fichier existe, son contenu est écrasé.
`a+b` ou `ab+`	Ouvre un fichier binaire en mode ajout pour lecture ou mise à jour à la fin du fichier. La fonction `open()` crée le fichier s'il n'existe pas.

TABLE 17.1 – Modes d'ouverture des fichiers

```
>>> MonFichier.read()/
# lit la totalité du fichier sous forme d'une chaîne de caractères
>>> MonFichier.readline()/
# lit la ligne suivante du fichier, sous forme d'une chaîne de caractères
>>> MonFichier.readlines()/
# lit toutes les lignes du fichier sous forme d'une liste de chaînes de caractères
>>> Monfichier.write('toto')/
# écrit la chaîne de caractère 'toto' dans le fichier
```

Pour écrire dans un fichier, on peut donc faire comme suit :

```
>>> f=open("tempo.txt",'w')
>>> f.write("Hello boy")
9
>>> f.close()
```

Pour lire le contenu d'une ligne, il suffit de lire cette ligne avec la fonction `readline()`, puis de séparer les éléments avec la méthode `split()` de `str`. Par exemple :

```
>>> g=open("/media/....votre-chemin/eleve-python/tmp.txt",'r')
>>> ligne=g.readline()
>>> print(ligne)
Hello boy
>>> h=ligne.split()
>>> print(h)
['Hello', 'boy']
>>> g.close() # referme le fichier
```

Sous *Windows*, il faut remplacer les signes "/" par "\\" !

La commande `help(file)` permet de lister l'aide sur l'utilisation des fichiers.

Une autre façon d'importer est d'utiliser la commande `with ... as` , par exemple :

Sous *Windows* :

```
with open("E:\\Info_marco\\eleve-python\\tmp.txt",'r') as i:
...     lignes=i.read()
```

Sous *Ubuntu* :

```
>>> with open("/media/marco2/113A293E768D5BA6/Info_marco/eleve-python/tmp.txt",'r')
    as i:
...     lignes=i.read()
```

Si vous voulez tester le programme tout prêt, vous pouvez utiliser le script suivant :

externe0.py

```
1  # -*- coding: utf-8 -*-
2
3  with open("tmp.txt",'r') as fich:
4      lignes=fich.readline()
5      print(lignes)
```

Cette méthode présente l'avantage (et l'inconvénient !) de fermer le fichier à la fin de l'indentation.

Lorsqu'on lit un fichier, s'il ne reste pas assez de caractères au fichier pour satisfaire la demande, la lecture s'arrête tout simplement à la fin du fichier.
Si la fin du fichier est déjà atteinte, `read()`, renvoie une chaîne vide.

D'autres exemples de manipulation de fichiers externes sont fournis, notamment pour obtenir des graphes.
Vous les trouverez page 339.

17.3 Encodage

Pour les entrées et sorties, il faut préciser sous quelle forme exacte les données sont attendues. De la même façon, il faut pouvoir choisir le format des données exportées.
Pour toutes ces entrées ou sorties de chaînes de caractères, nous considérerons qu'il s'agit concrètement de séquences d'octets.

Python 3 utilise désormais le type de données bytes, spécifiquement conçu pour traiter les séquences (ou chaînes) d'octets. Les données de type *bytes* sont très similaires aux données de type *string*, à la différence près que ce sont strictement des séquences d'octets, et non des séquences de caractères !

Les caractères peuvent être encodés en octets, et les octets décodés en caractères, mais pas de manière univoque : puisqu'il existe plusieurs normes d'encodage et de décodage, la même chaîne *string* peut être convertie en plusieurs chaînes *bytes* différentes.

Par exemple, si on souhaite lire le fichier *tmp0.txt* et si on fait :

```
g=open("tmp0.txt", 'r')
lignes_g=g.readline()
print(lignes_g)
print(type(lignes_g))
g.close()
```

Un message d'erreur peut apparaître car le fichier est constitués d'octets, donc de type *bytes*.

En utilisant à la première ligne :

```
g=open("tmp0.txt", 'rb')
```

Le même problème apparaît.

En procédant ainsi, nous ne récupérons donc pas notre chaîne de caractères initiale, mais bien sa traduction concrète en octets, dans une donnée de type *bytes*. De façon simpliste, ce qui pose problème, ce sont les caractères accentués ou "exotiques".

Pour enregistrer ou lire une séquence d'octets (donc de type *bytes*), il faut toujours ouvrir le fichier en mode binaire, en utilisant l'argument "wb" ou "rb" au lieu de "w" ou "r" dans l'instruction `open()`. Dans ce cas, le fichier est écrit ou lu sous forme d'octets (classe *bytes*)

Si l'on souhaite lire ou écrire des chaînes avec accent, l'encodage sera extrêmement important. La norme d'encodage *utf-8* est celle qui est la plus utilisée pour différentes raisons.

Si on exécute le programme suivant, le codage est automatiquement fait en *utf-8* :

externe1.py

```
1  # -*- coding: utf-8 -*-
2
3  # ouvre le fichier bibi1.txt
4  # le crée si besoin
5  f = open('tmp0.txt','w')
6
7  # écrit dedans
8  f.write("""Le Cancre
9  Il dit non avec la tête
10 Mais il dit oui avec le coeur
11 Il dit oui à ce qu'il aime
12 Il dit non au professeur
13 Il est debout
14 On le questionne
15 Et tous les problèmes sont posés
16 Soudain le fou rire le prend
17 Et il efface tout
18 Les chiffres et les mots
19 Les dates et les noms
20 Les phrases et les pièges
21 Et malgré les menaces du maître
22 Sous les huées des enfants prodiges
23 Avec des craies de toutes les couleurs
24 Sur le tableau noir du malheur
25 Il dessine le visage du bonheur.
26 Jacques Prévert
27 """)
28
29 f.close()
30
31 g0=open("tmp0.txt",'r')
32 # ouvre le fichier tmp0.txt en mode lecture
33 # g1 = g0.read()
34 # print(g1)
35
36 # g2 = g0.readline()
37 # print(g2)
38 # g3 = g0.readlines()
39 # print(g3)
40 g4=g0.read(15)
41 print(g4)
42 g5=g0.read(52)
43 print(g5)
44 g6=g0.read(32)
```

```
45  print(g6)
```

On obtient alors :

```
Le Cancre
Il di
t non avec la tête
Mais il dit oui avec le coeur
Il
dit oui à ce qu'il aime
Il dit n
```

Explication :

```
g4=g0.read(15)
print(g4)
g5=g0.read(52)
print(g5)
g6=g0.read(32)
print(g6)
```

g4 lit les 15 premiers caractères, g5 lit les 52 suivants et g6 les 32 qui se trouvent à la suite.

⇒ **Activité 17. 91**

Écrivez un programme qui ouvre le fichier nommé *bibi1.txt* et qui affiche à l'écran son contenu.
Il ajoutera au texte précédent la chaîne " et à toutes" et enregistrera le résultat dans un fichier *bibi3.txt*.

Programme *write2.py* :

⇒ **Activité 17. 92**

À l'aide d'une boucle, écrivez un programme qui remplace dans les fichiers *bibi1.txt*, *bibi2.txt* et *bibi3.txt* créés en activité, la chaîne "Bonjour" par "Au revoir".
Vous pourrez utiliser le mode `r+` et la méthode `replace`.

Programme *write3.py* :

18

Bibliothèques : généralités

18.1 Importation

Des nombreuses extensions du langage sont disponibles dans des bibliothèques *Python*. On les appelle également modules ou librairies. Pour y accéder, on peut faire :

```
import module # par exemple import math
```

Pour utiliser une fonction `maFonction` définie dans *module*, on peut taper :

```
module.maFonction() # par exemple math.sin(2)
```

On peut également renommer le module alors qu'on l'importe :

```
import module as m # par exemple import module as mat
m.maFonction()             # mat.sin(2)
```

Il est possible d'importer uniquement certaines fonctions ou méthodes et de rendre leur usage transparent :

```
from module import maFonction # ex : from math import sin
maFonction()                  # sin(2)
```

On peut aussi tout importer par :

```
from module import *
```

Pour utiliser la fonction, il suffit alors de faire `maFonction()`.
Cette dernière méthode d'importation présente l'avantage d'alléger l'écriture du code. Elle présente l'inconvénient (surtout dans sa dernière forme, celle qui importe toutes les fonctions d'un module) d'encombrer l'espace de noms courants. Il se pourrait alors que certaines fonctions importées aient le même nom que celui d'une variable définie par vous-même, ou encore le même nom qu'une fonction importée depuis un autre module. Si cela se produit, l'un des deux noms en conflit n'est évidemment plus accessible.

Dans les programmes d'une certaine importance, qui font appel à un grand nombre de modules d'origines diverses, il sera donc toujours préférable de privilégier la première méthode, c'est-à-dire celle qui utilise des noms pleinement qualifiés.

On fait généralement exception à cette règle dans le cas particulier du module *tkinter*, parce que les fonctions qu'il contient sont très sollicitées (dès lors que l'on décide d'utiliser ce module).

18.2 Différentes bibliothèques

On pourra se reporter aux sites suivants :

- *Python Standard Library* : http://docs.python.org/3.3/library/index.html
- Index des librairies Python : https://pypi.python.org/pypi

Des exemples de bibliothèques ou modules :

- *random* : http://docs.python.org/3/library/random.html
- *SciPy* : http://www.scipy.org/
- *SymPy* : http://sympy.org/fr/index.html
- *NumPy* : http://www.numpy.org/
- *Matplotlib* / Pylab : http://matplotlib.org/
- *sys* : http://docs.python.org/3/library/sys.html
- *re* : http://docs.python.org/3/library/re.html
- *os* : http://docs.python.org/3/library/os.html
- *math* : http://docs.python.org/3/library/math.html
- *cmath* : http://docs.python.org/3/library/cmath.html
- *easyguy* : http://easygui.sourceforge.net/tutorial/gallery.html
- *tkinter* : http://docs.python.org/3/library/tkinter.html
- *PIL* : http://jlbicquelet.free.fr/scripts/python/pil/pil.php
- *CoolProp* : http://www.coolprop.org/coolprop/wrappers/Python/index.html
- *openpyxl*, *xlrd*, *xlwt*, ... : http://www.python-excel.org/
- *csv* : https://docs.python.org/3.3/library/csv.html
- ...

Nous étudierons certains modules dans les chapitres suivants. Un petit aperçu des possibilités offertes par *Python* est présenté ci-dessous :

Les expressions régulières et l'interrogation du web étendent ce qui a été fait avec les chaînes de caractères et les fichiers texte. La bibliothèque *pickle* permet de sauvegarder aisément toutes les informations utilisées par un programme *Python*. La bibliothèque *pyodbc* permet d'écrire un client pour une base de données.

Il en existe bien d'autres : *xlrd* et *csv* permettent de lire et d'écrire dans des fichiers *Excel* et *csv*. *pygame* permet d'écrire des jeux (ou plus généralement d'interagir en temps réel avec l'utilisateur). *pyopengl* permet de créer des scènes en $3D$ (connaître le *C* aide à comprendre la logique de sa syntaxe). *socket* permet de créer une connection entre deux programmes sur deux machines distantes via le protocole TCP/IP, fournissant la base d'une architecture client/serveur. *ctypes* et *scipy.weave* permettent de lier *C* et *Python* pour accélérer les fonctions les plus utilisées et les plus gourmandes en temps de calcul. *cv* est le port sous *Python* d'*OpenCV* : il permet l'acquisition par caméra et le traitement d'image ; il s'interface aisément avec *numpy* et *PIL*. *unittest* permet de formaliser les batteries de tests unitaires pour garantir [1] que le programme fonctionne ou au moins que son évolution n'entraîne pas de régression.

18.2.1 Tirages aléatoires et statistiques

Le module *random* permet des tirages aléatoires selon une grande variété de lois. Il est possible de faire coexister plusieurs générateurs pseudo-aléatoires dans un même programme.

Par défaut, la graine est initialisée avec l'horloge du système.

Quelques fonctions :

- `random()` : tirage aléatoire selon une loi uniforme sur $[0,1]$,

1. Autant que possible : de façon générale, prouver le bon fonctionnement d'un programme est indécidable.

- `gauss(m, s)` : tirage aléatoire selon une loi normale d'espérance m et de variance s^2,
- `expovariate(a)` : tirage aléatoire selon une loi exponentielle d'espérance $1/a$,
- `randint(a,b)` : tirage d'un entier aléatoire entre a et b, bornes inclues,
- `shuffle(x)` : mélange la liste fournie en paramètre,
- `seed(s)` : utilise `s` comme nouvelle graine. Cela permet de jouer plusieurs fois la même séquence pseudo-aléatoire,
- `choice(x)` : sélectionne un élément de `x` au hasard (loi équiprobable),
- `sample(x, k)` : sélectionne k éléments de `x`.

Voici un programme qui retourne 50 entiers (séparés par un espace) pris aléatoirement entre 1 et 20.

random-choix.py

```python
# -*- coding: utf-8 -*-

import random

a=range(1,20)
for i in range(50):
    choix = random.choice(a)
    print(choix,end=' ')
```

18.2.2 Traitement d'images

On pourra citer notamment la bibliothèque *PIL* qui peut effectuer un grand nombre d'opérations sur les images. On en verra des exemples en deuxième année : c'est l'objet du chapitre .

18.2.3 Interrogation du web et format HTML

Les modules *urllib* et *urllib2* permettent d'accéder à des pages web.

La fonction `urlopen(url)` prend comme paramètre la chaîne de caractère contenant l'url demandée (y compris "http ://") et renvoie la page web correspondante. Il convient de la lire comme un ensemble de lignes.

exurl.py

```python
# -*- coding: utf-8 -*-

import urllib.request
with urllib.request.urlopen('http://python.org/') as response:
   html = response.read()
   print(html)
```

La page web est livrée avec les balises HTML. Pour exploiter la structure du code HTML, on peut utiliser un parser, comme par exemple *Beautifulsoup*. Vous pourrez touver des infos ici par exemple : `http://www.crummy.com/software`

18.2.4 Le module *subprocess*

Voici ci un exemple de programme utilisant *subprocess* qui télécharge tous les fichiers possédant une extension *.exe* et *.pdf*. Les fichiers correspondants sont téléchargés dans le dossier courant.

aspi-total.py

```
1  # -*- coding: utf-8 -*-
2
3  import subprocess
4  # adresse url du site
5  url="http://beos.prepas.org/?q=Epreuve%20Orale%201130"
6
7  # téléchargement dans le répertoire courant
8  subprocess.call(["wget", "-r", "-np", "-nd",\
9  "-A.tex", "-A.pdf", url])
```

18.2.5 Le module *sys*

Le module *sys* donne accès à l'environnement du script et sert à récupérer les arguments de la ligne de commande. Notamment :

- `argv` : éléments de la ligne de commande. `argv[0]` est le nom du script, et les éléments suivants contiennent ses options et paramètres. Chaque élément est une chaîne de caractère, de type `str` : si l'argument souhaité est numérique, il convient de le transtyper par `k = int(argv[1])` ou `x = float(argv[1])`.
- `stdin` : entrée standard, c'est-à-dire le clavier par défaut.
- `stdout` : sortie standard, c'est-à-dire le terminal d'où est exécuté le script par défaut.
- `stderr` : sortie d'erreur.

somme.py

```
1  # -*- coding: utf-8 -*-
2
3  import sys
4
5  s = 0
6  for x in sys.argv[1:]:
7      s += float(x)
8  print ("Total :",s)
```

Le script précédent s'appelant *somme.py*, un appel à :
`python somme.py 31 -9.5 20.5`
provoquera l'affichage de :

```
Total : 42.0
```

18.2.6 Le module *os*

Le module *os* permet d'interagir avec le système. Il est possible de lister des répertoires, les fichiers, de les effacer, de les déplacer, ...

Un exemple ci-dessous :

remove-csv.py

```
1  # -*- coding: utf-8 -*-
2
3  import os
```

```
4  import os.path
5  import fnmatch
6  # Liste des répertoires source
7  rep_source=[]
8  rep_source.append('/media/marco3/Données 1,9To/TSI_14-15/fichiers-
                                          python/')
9
10 # Liste des sous-répertoires à exclure
11 rep_exclus = []
12 rep_exclus.append('exclu')
13
14 type_fichier = ['*.csv']
15
16 # compteur fichiers supprimés
17 cpt=0
18
19 for repertoire in rep_source:
20     for root, repertoires, fichiers in os.walk(repertoire):
21         for rep in repertoires:
22             if rep in rep_exclus:
23                 repertoires.remove(rep)
24
25         # Parcours récursif des types
26         for chaque_type in type_fichier:
27             # Parcours récursif des répertoires
28             for fichier in fnmatch.filter(fichiers, chaque_type):
29                 # affiche les fichiers supprimés
30                 print(fichier, ' supprimé')
31                 #efface les fichiers
32                 os.remove(os.path.join(root, fichier))
33                 # incrémente le compteur de fichiers supprimés
34                 cpt+=1
35
36 print(cpt, ' fichiers supprimé(s) !')
```

Ce programme permet de supprimer tous les fichiers ayant l'extension *csv* dans `rep_source` mais pas dans `rep_exclus`.

Un autre code qui permet de lister de façon récursive les fichiers d'un répertoire possédant l'extension *csv* :

liste-rep.py

```
1  #-*- coding: utf8 -*-
2
3  import os
4
5  def liste_fich(repertoire):
6      for chemin, rep, fichiers in os.walk(repertoire):
7          for nom_fich in  fichiers :
8              if nom_fich.endswith(".csv"):
9                  print(os.path.join(chemin, nom_fich))
10
11 liste_fich("/media/marco3/Données 1,9To/TSI_14-15/")
```

18.2.7 Le module *tkinter*

Le module *tkinter* de *Python* permet de créer des interfaces graphiques.

De nombreux composants graphiques (appelés "widgets") sont disponibles : fenêtres, boutons, cases à cocher, étiquettes, zones de texte, menus, zones graphiques, cadres, ...

De nombreux événements peuvent intervenir : clic sur la souris, déplacement de la souris, appui sur une touche du clavier, top d'horloge, ...

2 exemples ici :

ouvrir-image.py

```
1  # -*- coding: utf-8 -*-
2
3  import tkinter as tk
4  from tkinter import messagebox
5  from tkinter.filedialog import askopenfilename
6  from PIL import Image, ImageTk
7  import time
8
9  message_date = "Nous sommes le ",time.strftime("%d/%m/%Y"),\
10 "L'heure actuelle est ",time.strftime("%H:%M:%S")
11
12 def openfile():
13     global photo
14     im = askopenfilename(filetypes=[('png files', '.png'), ('jpg
                                        files', '.jpg')])
15     image = Image.open(im)
16     photo = ImageTk.PhotoImage(image)
17     canvas.itemconfig(cimage, image=photo)
18
19 def Aide():
20     messagebox.showinfo("Aide","Débrouillez-vous !")
21
22 def Info():
23     messagebox.showinfo("date",message_date)
24
25 maFenetre = tk.Tk()
26 canvas = tk.Canvas(maFenetre)
27 cimage = canvas.create_image(0, 0, anchor=tk.NW, image="")
28 canvas.pack()
29 ouvrir = tk.Button(maFenetre, text="Ouvrir",bg="red",fg="white",
                                        command=openfile)
30 fermer = tk.Button(maFenetre,text="Quitter",bg="blue",fg="white",
                                        command=maFenetre.destroy)
31 help_me = tk.Button(maFenetre, text="Aide",bg="yellow",fg="black",
                                        command=Aide)
32 interr = tk.Button(maFenetre,text=" ? ",command=Info)
33 ouvrir.pack(side=tk.LEFT)
34 fermer.pack(side=tk.LEFT)
35 help_me.pack(side=tk.LEFT)
36 interr.pack(side=tk.LEFT)
37 maFenetre.mainloop()
```

et :

calcul.py

```
1  # -*- coding: utf-8 -*-
2
3  from tkinter import *
4
5  fenetre = Tk()
6  fenetre.title("Conversions pieds - mètres")
7
8  def convert():
9      #Récupération des variables
```

```
10      val_pied = float(pieds.get())
11      val_metre = val_pied*0.3048
12      sortie.configure(text = val_metre)
13
14  # première rangée
15  Label(fenetre, text="Valeur en pieds : ").grid(row=0, column=0)
16  pieds=Entry(fenetre)
17  pieds.grid(row=0, column=1)
18
19  # deuxième rangée
20  Label(fenetre, text="Valeurs en mètres : ").grid(row=1, column=0)
21  metres=Label()
22  metres.grid(row=1, column=1)
23  sortie = Label(fenetre)
24  sortie.grid(row=1, column=1)
25
26  Button(fenetre,text='Calculer',command=convert).grid(row=1 , column=2
                                                         )
27
28  # quatrième rangée
29  Button(fenetre,text='Quitter',command=fenetre.destroy).grid(row=3,
                                                        column=1)
30
31  fenetre.mainloop()
```

19

Gestion du temps

19.1 Les modules *time* et *timeit*

Les modules *time* et *timeit* permettent de gérer le temps.

19.1.1 *time*

Le module time permet de mesurer le temps écoulé et le temps CPU (le temps passé par un processus en mémoire centrale) :

`time()` mesure le temps ordinaire, celui de l'horloge. Il s'agit d'un réel (`float`) qui représente le temps écoulé (en secondes) depuis le 1er janvier 1970. La valeur retournée représente le temps local (et non le temps UTC/GMT). Cette fonction est fort utile (tout comme *clock*) pour évaluer des différences de temps.

`clock()` retourne la durée en secondes CPU consommée par le processus depuis le début de son exécution.

`sleep(sec)` effectue une pause de sec secondes. Cela peut être un réel.

`asctime()` (signifiant "Ascii Time") convertit le temps local en string. *asctime(x)* convertit un time tuple *x* en string.

```
>>> import time
>>> time.time()
1377038452.9047775
>>> time.sleep(2)
```

19.1.2 *timeit*

Le module *timeit()* :

- lance une fois l'expression fournie en deuxième argument du constructeur `Timer` : c'est la partie configuration,
- exécute le premier argument du constructeur 1000000 (1 million de) fois de suite,
- retourne la moyenne des exécutions.

On peut spécifier un autre nombre d'exécutions en argument de *timeit*, ce qui donne, avec le nombre 3 :

```
>>> import timeit
>>> t = timeit.Timer("print ('exécution principale')","print('configuration')")
>>> print (t.timeit(3))
configuration
éxécution principale
éxécution principale
éxécution principale
3.558699995664938e-05
```

On peut encore répéter cela plusieurs fois, avec la méthode `repeat` et récupérer la liste des temps d'exécution :

```
>>> print(t.repeat(2,3))
configuration
éxécution principale
éxécution principale
éxécution principale
configuration
éxécution principale
éxécution principale
éxécution principale
[3.2839999676070875e-05, 2.5809999897319358e-05]
```

On peut tester la durée d'exécution du programme suivant (*time-fibo.py*) qui calcule les 10 premiers termes de la suite de Fibonacci.

time-fibo.py

```
1  # -*- coding: utf-8 -*-
2
3  import timeit
4
5  t = timeit.Timer('''
6  c=a+b
7  a=b
8  b=c
9  print (c)''','a,b=1,1')
10
11 print(t.timeit(10))
```

qui donne :

```
2
3
5
8
13
21
34
55
89
144
6.713000016134174e-05
```

Vous pourrez par exemple tester le programme suivant :

time-test.py

```
1  # -*- coding: utf-8 -*-
2
3  from math import sin
4
```

```
5  ########################
6  # avec time.time()
7  ########################
8  import time
9
10 t0 = time.time()
11
12 for x in range(-4, 5):
13     try:
14         print("\nx = {:.1f} donne\
15          sin x / x = {:.3f}".format(x,sin(x)/x))
16     except ZeroDivisionError: # exception
17         print("\nx = 0.0 donne\
18          sin x /x = 1.000") # exception pour x = 0
19
20 t1 = time.time()
21
22 duree1 = t1-t0
23 print("\nduree1 = ",duree1)
24
25 ########################
26 # avec timeit.Timer
27 ########################
28 import timeit
29
30 def temps_execution():
31     t = timeit.Timer("tryexcept()", "from __main__ import tryexcept")
32     return t.timeit(2)
33
34 def tryexcept():
35     for x in range(-4, 5):
36         try:
37             print("\nx = {:.1f} donne\
38             sin x / x = {:.3f}".format(x,sin(x)/x))
39         except ZeroDivisionError: # exception
40             print("\nx = 0.0 donne\
41             sin x /x = 1.000") # exception pour x = 0
42
43 print("temps_execution = ",temps_execution())
44
45 ########################
46 # avec timeit et repeat
47 ########################
48 def expo(x,n):
49     resultat = 1
50     for i in range(n):
51         resultat = resultat*x
52     return resultat
53
54 duree2 = timeit.Timer("expo(3,1000)",\
55 "from __main__ import expo")
56 print("\nduree2 = ",duree2.timeit(100))
57 # processus lancé 100 fois
58
59 print("\nduree3 = ",duree2.timeit(10000))
60 # processus lancé 10 000 fois
61 # attention, c'est 1 000 000 par défaut
62
63 print("\nduree4 = ",duree2.repeat(2,100))
64 # processus lancé 100 fois et bloc 2 fois
```

```
65 # attention, c'est 1 000 000 par défaut et 3
66 print("\nminimum des 2 essais = ",min(duree2.repeat(2,100)))
67
68 ########################
69 # avec time.clock()
70 ########################
71 t1 = time.clock()
72 expo(3,10000)
73 t2 = time.clock()
74 duree1=t2-t1
75 print("\nduree1 CPU = ",duree1)
76
77 t1 = time.clock()
78 print("\nexp(10000) = ", expo(3,10000))
79 t2 = time.clock()
80 duree2=t2-t1
81 print("\nduree2 CPU = ",duree2)
```

ou plus simplement le programme :

time-while.py

```
1  # -*- coding: utf-8 -*-
2
3  import time
4
5  compteur = 0
6  t1 = time.clock()
7
8  while True:
9      compteur += 1
10     if compteur == 1000000:
11         break
12
13 print(time.clock() - t1)
14
15 compteur = 0
16 t1 = time.clock()
17
18 while 1:
19     compteur += 1
20     if compteur == 1000000:
21         break
22
23 print(time.clock() - t1)
```

19.1.3 Test de programme externe

Si vous avez créé un programme *toto.py*, dont vous voulez calculer le temps d'exécution, alors lancez python, et procédez ainsi :

```
>>> import timeit
>>> t=timeit.Timer('import toto','')
>>> t.timeit()
```

19.1.4 Différents programmes

Des exemples utilisant *time* et *timeit* ci-dessous. Si vous avez besoin de comparer des temps d'exécution de fonctions, de programmes, de déclencher des chronomètres, vous pourrez vous reporter aux exemples suivants :

Le programme :

optimisation3.py

```
1  # -*- coding: utf-8 -*-
2
3  import timeit
4
5  def simplissime(x):
6      return x
7
8  timer = timeit.Timer(stmt='simplissime(3)', setup='from __main__
                                        import simplissime')
9  print (timer.timeit(number=int(1e7)))
```

donne :

```
0.5282797740001115
```

Le programme :

optimisation7.py

```
1  # -*- coding: utf-8 -*-
2
3  import timeit
4
5  variable = 'a = 2.0'
6  instruction = 'b = a**2'
7
8  # éxécute variable et instruction 10 fois
9  duree1 = timeit.repeat(instruction, variable, repeat = 10, number = 1
                                        )
10 print(duree1)
11
12 print()
13 # éxécute variable deux fois et instruction 10 fois
14 duree2 = timeit.repeat(instruction, variable, repeat = 2, number = 10
                                        )
15 print(duree2)
```

donne :

```
[1.0349000149290077e-05, 6.379996193572879e-07, 1.93000232684426e-07, 1.
    3599992598756216e-07, 1.3500039131031372e-07,
1.580001480760984e-07, 1.6499961930094287e-07, 1.339999471383635e-07, 1.
    3299995771376416e-07, 1.369999154121615e-07]

[1.788999725249596e-06, 8.839997462928295e-07]
```

Le module *time* permet de comparer des temps d'exécution. Par exemple, si on souhaite comparer les 4 fonctions suivantes du programme *stringoptim.py* :

stringoptim.py

```python
# -*- coding: utf-8 -*-

chaine1 = ["tagada"]*10 + ["tsouintsouin"]*10

def ex1():
    chaine_finale = []
    for mot in chaine1:
        if not "tsouin" in mot:
            chaine_finale.append(mot)
    return "".join(chaine_finale)

def ex2():
    chaine_finale = []
    app = chaine_finale.append
    for mot in chaine1:
        if not "tsouin" in mot:
            app(mot)
    return "".join(chaine_finale)

def ex3():
    chaine_finale = ""
    for mot in chaine1:
        if mot[:5] == "tagad":
            chaine_finale += mot
    return chaine_finale

def ex4():
    chaine_finale = ""
    for mot in chaine1:
        if mot.startswith("tagad"):
            chaine_finale += mot
    return chaine_finale

chaine2 = ["tagada"]*10000000 + ["tsouintsouin"]*100000

def ex10():
    chaine_finale = []
    for mot in chaine2:
        if not "tsouin" in mot:
            chaine_finale.append(mot)
    return "".join(chaine_finale)

def ex20():
    chaine_finale = []
    app = chaine_finale.append
    for mot in chaine2:
        if not "tsouin" in mot:
            app(mot)
    return "".join(chaine_finale)

def ex30():
    chaine_finale = ""
    for mot in chaine2:
        if mot[:6] == "tagada":
            chaine_finale += mot
    return chaine_finale

def ex40():
```

```
59      chaine_finale = ""
60      for mot in chaine2:
61          if mot.startswith("tagada"):
62              chaine_finale += mot
63      return chaine_finale
```

On peut alors utiliser le programme suivant qui va comparer leur temps d'exécution.

optimisationext.py

```
1  # -*- coding: utf-8 -*-
2
3  import timeit
4
5  def footime(s, n=1):
6      t = timeit.Timer(s, 'import stringoptim')
7      time = t.timeit(n)
8      print (time)
9
10 footime('stringoptim.ex1()', n=5)
11 footime('stringoptim.ex2()', n=5)
12 footime('stringoptim.ex3()', n=5)
13 footime('stringoptim.ex4()', n=5)
14 footime('stringoptim.ex10()', n=1)
15 footime('stringoptim.ex20()', n=1)
16 footime('stringoptim.ex30()', n=1)
17 footime('stringoptim.ex40()', n=1)
```

On obtient alors :

```
1.4151000414130976e-05
8.923000223148847e-06
1.4303999705589376e-05
1.9336000150360633e-05
0.7544117070001448
0.5766118030001053
1.2779790929998853
1.780076218999966
```

On peut également tester un programme externe, par exemple le programme aperçu dans la section 6.9 page 80 :

try_except.py

```
1  # -*- coding: utf-8 -*-
2
3  from math import sin
4  for x in range(-3, 4):
5      try:
6          print("\nx = {:.1f} donne\
7           sin(x)/x = {:.3f}".format(x,sin(x)/x))
8      except ZeroDivisionError as erreur: # exception
9          print("\nx = 0.0 donne\
10          sin(x)/x = 1.000") # exception pour x = 0
11         print("Cette erreur a été évitée : ",erreur)
12
13 for x in range(-3, 4):
14     print("\nx = {:.1f} donne\
15     sin(x)/x = {:.3f}".format(x,sin(x)/x))
```

Le programme ci-dessous est très simple :

testtry_except.py

```
1  # -*- coding: utf-8 -*-
2
3  import timeit
4
5  t=timeit.Timer('import try_except','')
6
7  print("\n",t.timeit())
```

et donne :

```
x = -3.0 donne          sin(x)/x = 0.047

x = -2.0 donne          sin(x)/x = 0.455

x = -1.0 donne          sin(x)/x = 0.841

x = 0.0 donne          sin(x)/x = 1.000
Cette erreur a été évitée :  float division by zero

x = 1.0 donne          sin(x)/x = 0.841

x = 2.0 donne          sin(x)/x = 0.455

x = 3.0 donne          sin(x)/x = 0.047

 0.23298875199998292
```

19.2 Le module *datetime*

Un exemple pour illustrer ce module :

```
>>> from datetime import date datetime
>>> date.today()
datetime.date(2013, 8, 30)
>>> maintenant=datetime.now()
>>> maintenant
datetime.datetime(2013, 8, 30, 0, 5, 54, 827840)
>>> maintenant.year
2013
>>> maintenant.month
8
>>> maintenant.day
30
>>> maintenant.hour
0
>>> maintenant.minute
5
>>> maintenant.second
54
>>> maintenant.microsecond
827840
```

19.3 Le module *calendar*

⇒ **Activité 19. 93**

Indiquez le rôle des instructions suivantes :

```
>>> import calendar
>>> calendar.mdays
[0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
>>> calendar.isleap(2014)
False
>>> calendar.weekday(2013,8,29)
3
```

20

La bibliothèque numpy

Il est impossible de donner une liste exhaustive de toutes les bibliothèques relatives à *Python*. Toutefois, pour les applications scientifiques, la bibliothèque *numpy* apparaît incontournable.

20.1 Importation de la bibliothèque *numpy*

Comme déjà vu précédemment, l'importation d'une bibliothèque se fait grâce à la commande `import`. Pour accéder à une fonction particulière de la bibliothèque, on peut taper :
`nom_de_la_bibliothèque.nom_de_la_fonction()`. Par exemple, pour la fonction `sin()` :

```
>>> import numpy
>>> numpy.sin(3.141592654)
-4.1020685703470686e-10
```

Pour éviter de retaper le nom complet de la bibliothèque (qui dans certain peut être assez long), on peut modifier le nom de la bibliothèque en utilisant le terme `as`. Par exemple, ici on renomme `numpy` par `np` :

```
>>> import numpy as np
>>> np.sin(3.141592654)
-4.1020685703470686e-10
```

On peut se passer complètement du rappel du nom de la bibliothèque, en spécifiant directement les fonctions qu'on désire importer avec le terme `from` :

```
>>> from numpy import sin
>>> sin(3.141592654)
-4.1020685703470686e-10
```

On peut même importer directement toutes les fonctions d'une bibliothèque avec le symbole $*$:

```
>>> from numpy import *
>>> sin(3.141592654)
-4.1020685703470686e-10
```

A priori, la dernière solution semble la plus simple car elle évite des écritures répétitives. D'ailleurs par défaut, la fenêtre console de *spyder* effectue directement l'importation de toutes les fonctions de la bibliothèque `numpy`. Ainsi, sous *spyder*, on peut sans `import` expliciter lancer la commande suivante :

```
>>> sin(1.5)
0.99749498660405445
```

Toutefois, comme déjà dit et d'une manière générale, l'import systématique de fonction est à déconseiller fortement pour deux raisons. La première est que l'on importe en général une grande quantité de fonctions dont on ne se sert pas réellement. La seconde est qu'en *Python*, on est amené importer à un nombre conséquent de bibliothèques dont certaines fonctions peuvent porter le même nom.

Par exemple, la bibliothèque *math* existe et possède également la fonction `sin()`. D'après la séquence de code suivante, les deux fonctions `sin()` semblent se comporter de la même façon sur un nombre, mais on constatera qu'elles sont très différentes dès lors qu'elles s'appliquent sur une liste :

```
>>> import numpy as np
>>> import math as m
>>> a=3.14
>>> np.sin(a)
0.0015926529164868282
>>> m.sin(a)
0.0015926529164868282
>>> b=[1.5,3.14]
>>> np.sin(b)
array([ 0.99749499,  0.00159265])
>>> m.sin(b)
Traceback (most recent call last)~:
    File "<stdin>", line 1, in <module>
TypeError~: a float is required
```

20.2 Les types sous Numpy

L'importation de la libraire *numpy* augmente le nombre des types de possible. En plus des entiers classiques (`int`), on peut considérer les entiers positifs (`uint`). On peut aussi spécifier le nombre de bits utilisé pour le codage. La liste des types devient alors : `bool`, `int`, `int8` (de -128 à 127), `int16` (de -32 768 à 32 767), `int32` (de -2 147 483 648 à 2 147 483 647), `int64` (de -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807), `uint8` (de 0 à 255), `uint16` (de 0 à 65 535), `uint32` (de 0 à 4 294 967 295), `uint64` (de 0 à 18 446 744 073 709 551 615), `float`, `float16`, `float32`, `float64`, `complex`, `complex64` et `complex128`.

20.3 Quelques fonctions numériques sous *numpy*

La listes des fonctions numériques définies dans *numpy* est extrêmement importante. On peut citer ici les fonctions les plus classiques (remarque : les termes entre "[" et "]" sont des paramètres optionnels).

20.3.1 Trigonométrie

`sin(x[, out])`	sinus
`cos(x[, out])`	cosinus
`tan(x[, out])`	tangente
`arcsin(x[, out])`	arcsinus
`arccos(x[, out])`	arccosinus
`arctan(x[, out])`	arctangente
`hypot(x1, x2[, out])`	hypothénuse en fonction des deux autres cotés
`arctan2(x1, x2[, out])`	arctangente du rapport x1/x2 (quadrant selon x1 et x2)
`degrees(x[, out])`	conversion de radians vers degrés
`radians(x[, out])`	conversion de degrés vers radians
`unwrap(p[, discont, axis])`	déroulement de la phase
`deg2rad(x[, out])`	conversion de degrés vers radians
`rad2deg(x[, out])`	conversion de radians vers degrés

TABLE 20.1 – Trigonométrie avec numpy

20.3.2 Trigonométrie hyperbolique

`sinh(x[, out])`	sinus hyperbolique
`cosh(x[, out])`	cosinus hyperbolique
`tanh(x[, out])`	tangente hyperbolique
`arcsinh(x[, out])`	arcsin hyperbolique
`arccosh(x[, out])`	arccosinus hyperbolique
`arctanh(x[, out])`	arcsinus hyperbolique

TABLE 20.2 – Trigonométrie hyperbolique avec numpy

20.3.3 Arrondis

`around(a[, decimals, out])`	arrondi à une décimale près
`rint(x[, out])`	entier le plus proche
`fix(x[, y])`	arrondi à 0 décimale
`floor(x[, out])`	arrondi à la valeur inférieure
`ceil(x[, out])`	arrondi à la valeur supérieure
`trunc(x[, out])`	arrondi à l'entier le plus proche de zéro

TABLE 20.3 – Arrondis avec Numpy

20.3.4 Exponentielles et logarithmes

`exp(x[, out])`	exponentielle
`expm1(x[, out])`	exponentielle moins 1
`exp2(x[, out])`	exponentielle de base 2
`log(x[, out])`	logarithme naturel
`log10(x[, out])`	logarithme de base 10
`log2(x[, out])`	logarithme de base 2
`log1p(x[, out])`	logarithme de $x+1$

TABLE 20.4 – Exponentielles et logarithmes avec Numpy

20.3.5 Fonctions spéciales

`i0(x)`	Bessel modifiée de première espèce d'ordre 0
`sinc(x)`	sinus cardinal

TABLE 20.5 – Fonctions spéciales avec Numpy

Beaucoup d'autres fonctions spéciales existent dans *numpy* mais elles appartiennent à des sous-librairies.

20.3.6 Autres fonctions

`signbit(x[, out])`	indication de la présence d'un bit de signe
`copysign(x1, x2[, out])`	copie du signe de x2 sur x1
`frexp(x[, out1, out2])`	décomposition $x = \text{out1} * 2^{\text{out2}}$
`ldexp(x1, x2[, out])`	calcul $\text{x1} * 2^{\text{x2}}$

TABLE 20.6 – Autres fonctions avec numpy

20.3.7 Arithmétique

`add(x1, x2[, out])`	addition
`reciprocal(x[, out])`	inversion (Attention cas entier)
`negative(x[, out])`	opposé
`multiply(x1, x2[, out])`	multiplication
`divide(x1, x2[, out])`	division
`power(x1, x2[, out])`	puissance $x1^{x2}$
`subtract(x1, x2[, out])`	soustraction
`true_divide(x1, x2[, out])`	vraie division (même entiers)
`floor_divide(x1, x2[, out])`	quotient division entière (même flottants)
`fmod(x1, x2[, out])`	reste de division entière
`mod(x1, x2[, out])`	reste de division entière
`modf(x[, out1, out2])`	partie décimale et partie entière
`remainder(x1, x2[, out])`	reste de division entière

TABLE 20.7 – Arithmétique avec numpy

20.3.8 Complexes

`angle(z[, deg])`	argument
`real(val)`	partie réelle
`imag(val)`	partie imaginaire
`conj(x[, out])`	complexe conjugué

TABLE 20.8 – Complexes avec numpy

20.3.9 Fonctions diverses

`sqrt(x[, out])`	racine carrée
`square(x[, out])`	mise au carré
`absolute(x[, out])`	valeur absolue
`fabs(x[, out])`	valeur absolue
`sign(x[, out])`	signe (-1,0,1)
`maximum(x1, x2[, out])`	maximum
`minimum(x1, x2[, out])`	minimum
`nan_to_num(x)`	remplacement de `nan` et de `inf`
`real_if_close(a[, tol])`	élimination des petites parties imaginaires

TABLE 20.9 – Fonctions diverses avec numpy

Remarque

`numpy.nan` (Not A Number) et `numpy.inf` (Infini) sont définis dans la bibliothèque *numpy*.

20.4 Les matrices sous *numpy*

20.4.1 Structures de base

En partant des types fondamentaux, il est possible de créer des matrices et des vecteurs en utilisant le concept de liste.

Une liste se crée en utilisant "[ ]" :

```
>>> a=[1.3,2.0,7.6]
>>> a[0]
1.3
>>> a[1]
2.0
>>> a[2]
7.6
```

On notera que pour une liste de taille n, l'indice varie de 0 à $n-1$.

Pour créer une matrice, il est possible de créer une liste de liste :

```
>>> b=[[1,2,3],[4,5,6],[7,8,9]]
>>> print (b)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> b[1]
[4, 5, 6]
>>> b[1][2]
6
```

Deux fonctions très utiles pour les listes de nombres :

- `range()` permet de générer une liste de nombres entiers avec un pas régulier. Tous les paramètres doivent être des entiers.
- `len()` donne la longueur d'une liste.

Ces deux fonctions peuvent donner lieu à la séquence suivante :

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(6,9)
[6, 7, 8]
>>> range(1.2,5)
Traceback (most recent call last)~:
    File "<stdin>", line 1, in <module>
TypeError~: range() integer start argument expected, got float.
>>> range(1,10,2)
[1, 3, 5, 7, 9]
>>> range(10,1,-2)
[10, 8, 6, 4, 2]
>>> a=range(4,8)
>>> len(a)
4
>>> b=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
>>> print (b)
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> len(b)
3
```

Quoi qu'il en soit, il ne sera pas possible de travailler sur des listes ou des listes de listes de façon algébrique. Par exemple, le produit direct de listes conduit à une erreur :

```
>>> a=[1.3,2.0,7.6]
>>> b=[[1,2,3],[4,5,6],[7,8,9]]
>>> print (b)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> b*a
Traceback (most recent call last)~:
    File "<stdin>", line 1, in <module>
TypeError~: can't multiply sequence by non-int of type 'list'
```

Pour réaliser des opérations algébriques, nous sommes condamnés à reprogrammer les opérations fondamentales de l'algèbre linéaire. En ajoutant le fait qu'en *Python* de base, les opérations arithmétiques et fonctions numériques sont limitées, l'emploi de bibliothèques mathématiques est indispensable pour des applications scientifiques.

20.4.2 Le type matrix sous *numpy*

En plus des types de nombre et des fonctions numériques vues précédemment, la bibliothèque *numpy* apporte également de nouvelles structures. En particulier, il existe la structure `matrix` qui n'est pas une simple liste. Ainsi, le calcul algébrique suivant :

$$\vec{u} = A \cdot \vec{v} \text{ avec } \vec{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \text{ et } A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

peut se traduire sous *Python* par la séquence :

```
>>> v=np.matrix([1,2,3])
>>> print (v)
matrix([[1, 2, 3]])
>>> A=np.matrix([[1,0,0],[0,-1,0],[0,0,2]])
>>> print (A)
matrix([[ 1,  0,  0],
        [ 0, -1,  0],
        [ 0,  0,  2]])
>>> A*v
Traceback (most recent call last)~:
    File "<stdin>", line 1, in <module>
    File "/usr/lib/python2.7/dist-packages/numpy/matrixlib/defmatrix.py", line
330, in __mul__
        return N.dot(self, asmatrix(other))
ValueError~: objects are not aligned
>>> vt=np.transpose(v)
>>> print (vt)
matrix([[1],
        [2],
        [3]])
>>> A*vt
matrix([[ 1],
        [-2],
        [ 6]])
```

L'avantage de la structure `matrix` de *numpy* est que l'opérateur $*$ a le même sens que la multiplication au sens des matrices et que les opérations algébriques deviennent transparentes.

Par ailleurs, il existe un grand nombre de fonctions applicables au type `matrix` : `transpose`, `trace`, `diagonal`, `reshape`, ...

Par exemple, pour transposer une matrice. On peut écrire sous *Python* :

```
>>> import numpy as np
>>> A=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> np.transpose(A)
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
```

Toutefois, nous développerons pas plus les traitements mathématiques avec le type `matrix` de *numpy*. En effet, ce type possède de sérieuses limitations. En particulier, il n'est pas possible de travailler en dimension supérieure à 2.

20.4.3 Le type `array` sous *numpy*

Le type `array` est plus général. Il permet de travailler en dimension quelconque et c'est la structure qui est privilégiée sous *numpy*.

Avec le type `array`, il est possible de réaliser quasiment les mêmes opérations que nous avons précédemment réalisées avec le type `matrix`. Toutefois, si nous réécrivons la même séquence que précédemment en utilisant le type `array`, nous noterons certaines différences :

```
>>> import numpy as np
>>> v=np.array([1,2,3])
>>> print (v)
[1 2 3]
>>> v
array([1, 2, 3])
>>> A=np.array([[1,0,0],[0,-1,0],[0,0,2]])
>>> print (A)
[[ 1  0  0]
 [ 0 -1  0]
 [ 0  0  2]]
>>> A
array([[ 1,  0,  0],
       [ 0, -1,  0],
       [ 0,  0,  2]])
>>> A*v
array([[ 1,  0,  0],
       [ 0, -2,  0],
       [ 0,  0,  6]])
>>> vt=np.transpose(v)
>>> A*vt
array([[ 1,  0,  0],
       [ 0, -2,  0],
       [ 0,  0,  6]])
```

20.5 Manipulation des matrices

À partir de maintenant nous travaillerons (sauf mention explicite) avec le type `array` de *numpy* et nous admettrons que l'importation de la bibliothèque *numpy* est réalisée par :

```
import numpy as np
```

20.5.1 Les bases

Les matrices de type `array` peuvent se construire à partir des listes de *Python*.

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Un `array` peut être multidimensionnel (ici dimension 3).

```
>>> a=np.array([[[1,2],[1,2]],[[1,2],[1,2]]])
>>> a
array([[[1, 2],
        [1, 2]],

       [[1, 2],
        [1, 2]]])
>>> type(a)
<type 'numpy.ndarray'>
>>> a[1,1,1]
2
```

On peut manipuler les matrices en travaillant sur les indices (par défaut les indices commencent à 0). Il est important de comprendre la notion d'égalité pour les matrices.

```
>>> a=np.array([1, 2, 3, 4, 5])
>>> b=a[1:3]
>>> b
array([2, 3])
>>> b[1]=0
>>> a # a est modifié
array([1, 2, 0, 4, 5])
```

La notion d'égalité est différent de la notion de copie (fonction `copy()`).

```
>>> a=np.array([1, 2, 3, 4, 5])
>>> b=np.copy(a[1:3])
>>> b
array([2, 3])
>>> b[1]=0
>>> a
array([1, 2, 3, 4, 5])
>>> b
array([2, 0])
```

Pour la copie, nous avons utilisé une fonction *numpy*. Il est possible de faire autrement car un `array` est une structure bien plus complexe que de simples listes. Un `array` possède en réalité une structure objet : en plus des données, à chaque 'array est associé un ensemble de fonctions (on parlera alors de méthodes) qu'il est possible d'appeler.

```
>>> a=np.array([1, 2, 3, 4, 5])
>>> b=a[1:3].copy()
>>> b[1]=0
>>> b
array([2, 0])
>>> a
array([1, 2, 3, 4, 5])
```

Manipulation sur les indices et fonction `arange()` :

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3] # [début:fin:pas]
array([2, 5, 8])
>>> a[2:8:3] # le dernier élément n'est pas inclus
array([2, 5])
>>> a[:5]    # le dernier élément n'est pas inclus
array([0, 1, 2, 3, 4])
>>> a[8:2:-3] # les pas négatifs sont autorisés ; l'ordre du tableau est inversé
array([8, 5])
```

Un pas négatif inversera l'ordre du tableau. Un indice négatif est considéré comme le nombre d'éléments précédant le dernier élément du tableau. Ainsi `a[-2::1]` comme `a[-2:]` extraient l'avant-dernier et le dernier élément.

Pour un tableau bi-dimensionnel, on peut bien sûr jouer avec les deux indices. La fonction `eye()` permet de créer des matrices avec des 1 sur la diagonale et des 0 ailleurs :

```
>>> a=np.eye(5,5)
>>> print (a)
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
>>> a[:3,:4]=4
>>> a[::3,::4]=5
>>> a[:,2]=6
>>> print (a)
[[ 5.  4.  6.  4.  5.]
 [ 4.  4.  6.  4.  0.]
 [ 4.  4.  6.  4.  0.]
 [ 5.  0.  6.  1.  5.]
 [ 0.  0.  6.  0.  1.]]
```

Les indices peuvent aussi être des listes. La fonction `reshape()` permet de modifier la taille des matrices :

```
>>> a=np.array([2,4,6,8],float)
>>> b=np.array([0,0,1,3,2,1],int)
>>> a[b]
array([ 2.,  2.,  4.,  8.,  6.,  4.])
>>> a=a.reshape(2,2)
>>> a
array([[ 2.,  4.],
       [ 6.,  8.]])
>>> b=np.array([0,0,1,1,0],int)
>>> c=np.array([0,1,1,1,1],int)
>>> a[b,c]
array([ 2.,  4.,  8.,  8.,  4.])
```

On peut aussi convertir une matrice de type `array` en type `matrix` :

```
>>> a=np.array([1, 2, 4])
>>> np.mat(a)
matrix([[1, 2, 4]])
```

20.5.2 Construction des matrices

Construction de matrices avec la fonction `arange()` :

```
>>> np.arange(10) # entiers (<10) en partant de 0
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2, 10, dtype=np.float) # flottants (<10) dont le premier est 2.
array([ 2., 3., 4., 5., 6., 7., 8., 9.])
>>> np.arange(2, 3, 0.1)  # pas non entier
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
```

Construction de matrices avec la fonction `linspace()` :

```
>>> np.linspace(1., 4., 6) # 6 flottants régulièrement répartis de 1. à 4. (inclus)
array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])
```

Utilisation de la fonction `reshape()` (Attention à respecter le nombre d'éléments) :

```
>>> a=np.arange(16)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a.reshape(4,4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>> a.reshape(2,8)
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15]])
```

Il existe aussi la fonction `flatten()` transforme une matrice en un vecteur.

Il est possible de concaténer avec la fonction `concatenate()` des matrices verticalement ou horizontale selon la valeur de `axis`.

```
>>> a=np.arange(4).reshape(2,2)
>>> b=4+np.arange(4).reshape(2,2)
>>> b
array([[4, 5],
       [6, 7]])
>>> c=(np.arange(4)+6)[newaxis,~:]
>>> np.concatenate((a,b))
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
>>> np.concatenate((a,b),axis=0)
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
>>> np.concatenate((a,b),axis=1)
array([[0, 1, 4, 5],
       [2, 3, 6, 7]])
>>> np.concatenate((c,c,np.concatenate((a,b),axis=1)))
array([[6, 7, 8, 9],
       [6, 7, 8, 9],
       [0, 1, 4, 5],
       [2, 3, 6, 7]])
```

La fonction `arange()` génère un vecteur (dimension 1) qui est différent d'une matrice à une ligne ou une colonne (dimension 2). Pour transformer un vecteur en matrice, il faut ajouter une dimension avec `np.newaxis` :

```
>>> c=(np.arange(4)+6)
>>> c
array([6, 7, 8, 9])
>>> c=(np.arange(4)+6)[np.newaxis,~:]
>>> c
array([[6, 7, 8, 9]])
```

Pour travailler sur les indices, il existe les fonctions `take()` et sa réciproque `put()` :

```
>>> a=np.arange(16).reshape(4,4)
>>> a.take([0,3],axis=1)
array([[ 0,  3],
       [ 4,  7],
       [ 8, 11],
       [12, 15]])
>>> a.take([0,3])
array([0, 3])
>>> a.take([0,3],axis=0)
array([[ 0,  1,  2,  3],
       [12, 13, 14, 15]])
>>> a.take([0,3,2,2,2],axis=0)
array([[ 0,  1,  2,  3],
       [12, 13, 14, 15],
       [ 8,  9, 10, 11],
       [ 8,  9, 10, 11],
       [ 8,  9, 10, 11]])
```

```
>>> a=np.array([0, 1, 2, 3, 4, 5],float)
>>> b=np.array([9,8,7],float)
>>> a.put([0,3],b)
>>> a
array([ 9.,  1.,  2.,  8.,  4.,  5.])
```

20.5.3 Matrices particulières

`zeros(n)`, `zeros((n,p))`	Vecteur ou matrice de 0
`ones(n)`, `ones((n,p))`	Vecteur ou matrice de 1
`eye(n)`, `eye(n,p)`	Diagonale de 1 et 0 ailleurs
`diag(v)`	Matrice dont la diagonale est le vecteur v
`diag(v,k)`	Matrice `diag(v)` mais diagonale décalée de k

TABLE 20.10 – Matrices particulières avec array de numpy

```
>>> v=np.ones(5)
>>> v
array([ 1.,  1.,  1.,  1.,  1.])
>>> np.diag(v,2)
array([[ 0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> np.diag(v,-1)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

On peut aussi générer des vecteurs ou des matrices aléatoires (distribution uniforme) de nombres entre 0 et 1 avec la fonction `rand()`. Mais cette fonction appartient à une sous bibliothèque *random*, que l'on peut appeler comme cela :

```
>>> np.random.rand(4)
array([ 0.67059949,  0.17947019,  0.7217041 ,  0.7823382 ])
>>> np.random.rand(4,2)
array([[ 0.17309785,  0.82626591],
       [ 0.7911254 ,  0.19542556],
       [ 0.45654595,  0.92389837],
       [ 0.74930928,  0.40855282]])
>>> from numpy.random import rand # alternative pour importer la fonction rand
>>> rand(2)
array([ 0.22220557,  0.02390764])
>>> rand(2,3)
array([[ 0.17719008,  0.78686815,  0.89788023],
       [ 0.00363013,  0.2147351 ,  0.47286765]])
```

20.5.4 Opérations

On peut additionner deux vecteurs ou deux matrices de même dimensions avec l'opérateur +. On peut aussi additionner ou multiplier un scalaire par un vecteur ou une matrice par exemple `2.+a` ou `2.*a`. De même `a**3` passe chaque élément de `a` au cube et `1/a` calcule l'inverse de chaque élément de `a` :

```
>>> a=np.random.rand(2,2)+np.ones((2,2))
>>> a
array([[ 1.59691955,  1.2573431 ],
       [ 1.17175816,  1.70118795]])
>>> 1/a
array([[ 0.62620562,  0.79532786],
       [ 0.85341842,  0.58782453]])
```

L'opérateur $*$ entre deux vecteurs ou deux matrices de même dimension effectue une multiplication terme à terme. Entre une matrice et un vecteur de dimension compatible, l'opérateur $*$ effectue une multiplication terme à terme sur chaque ligne et non une multiplication algébrique (contrairement au type `matrix`) :

```
>>> a=3.*np.ones((2,2))
>>> a*a
array([[ 9.,  9.],
       [ 9.,  9.]])
>>> v=np.arange(1,3)
>>> v
array([1, 2])
>>> a*v
array([[ 3.,  6.],
       [ 3.,  6.]])
```

Le produit algébrique se réalise avec la fonction `dot()` :

```
>>> a=np.arange(4).reshape(2,2)
>>> v=np.array([-3,2])
>>> np.dot(a,v)
array([2, 0])
>>> np.dot(v,a)
array([4, 3])
>>> w=np.concatenate((v,v))
>>> w
array([-3,  2, -3,  2])
>>> dot(a,w)
Traceback (most recent call last)~:
    File "<stdin>", line 1, in <module>
ValueError~: objects are not aligned
```

On peut aussi réaliser le produit scalaire de 2 vecteurs avec la fonction `vdot()`. Il existe aussi le produit de Kronecker avec `kron()` :

```
>>> a=np.arange(4,dtype=float).reshape(2,2)
>>> b=np.arange(5,dtype=float)
>>> print a,b
[[ 0.  1.]
 [ 2.  3.]] [ 0.  1.  2.  3.  4.]
>>> np.kron(a,b)
array([[  0.,   0.,   0.,   0.,   0.,   0.,   1.,   2.,   3.,   4.],
       [  0.,   2.,   4.,   6.,   8.,   0.,   3.,   6.,   9.,  12.]])
>>> np.kron(b,a)
array([[  0.,   0.,   0.,   1.,   0.,   2.,   0.,   3.,   0.,   4.],
       [  0.,   0.,   2.,   3.,   4.,   6.,   6.,   9.,   8.,  12.]])
>>> a=np.arange(5,dtype=float)
>>> b=np.ones((1,5))
>>> np.kron(a,b.transpose())
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.]])
```

La fonction `rank()` calcule le rang d'une matrice.

20.5.5 Sous-bibliothèque linalg

La sous-bilbiothèque *linalg* permet l'inversion (`inv()`), la résolution de systèmes linéaires (`solve()`), le calcul du déterminant (`det()`) et le calcul des vecteurs et valeurs propres (`eig()`) :

```
>>> a=np.array([2,4,6,8],float).reshape(2,2)
>>> np.linalg.inv(a)
array([[-1.  ,  0.5 ],
       [ 0.75, -0.25]])
```

```
>>> a=np.array([2,4,6,8],float).reshape(2,2)
>>> b=np.array([1, 4],float)
>>> np.linalg.solve(a,b)
array([ 1.  , -0.25])
>>> b=np.array([[1, 1],[4, -4]],float)
>>> np.linalg.solve(a,b)
array([[ 1.  , -3.  ],
       [-0.25,  1.75]])
```

```
>>> a=np.array([2,4,6,8],float).reshape(2,2)
>>> np.linalg.eig(a)
(array([ -0.74456265,  10.74456265]), array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]]))
```

20.5.6 Fonctions numériques de *numpy* sur les matrices

Toutes les fonctions numériques de *numpy* s'appliquent aussi à des vecteurs ou des matrices élément par élément :

```
>>> a=np.arange(5)
>>> a
array([0, 1, 2, 3, 4])
>>> np.square(a)
array([ 0,  1,  4,  9, 16])
```

21

La bibliothèque Matplotlib

21.1 Tracer des courbes

Il existe différentes bibliothèques qui permettent de tracer des courbes. Ces bibliothèques sont plus ou moins complètes, se caractérisent par un certain niveau de facilité à la mise en œvre et ont une qualité graphique donnée. Parmi toutes celles possibles, la bibliothèque *matplotlib* présente un excellent compromis et est d'usage extrêmement fréquent.

Sur le web, il est très simple de trouver des présentations de l'utilisation de la bibliothèque *matplotlib* avec les codes en open source.

Vous trouverez aussi de très belles choses ici :
http://matplotlib.org/index.html

21.2 Premiers graphes

Voici un premier exemple très simple utilisant la sous-bibliothèque *pyplot* qui permet de tracer une courbe et ouvre une fenêtre graphique :

Programme :

mpl-1.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  plt.plot([1,2,3,4])
7  plt.xlabel('Abscisses') # nom des abscisses
8  plt.ylabel('Ordonnées') # nom des ordonnées
9
10 plt.savefig("mpl-1.eps",dpi=200)
11 plt.show()
```

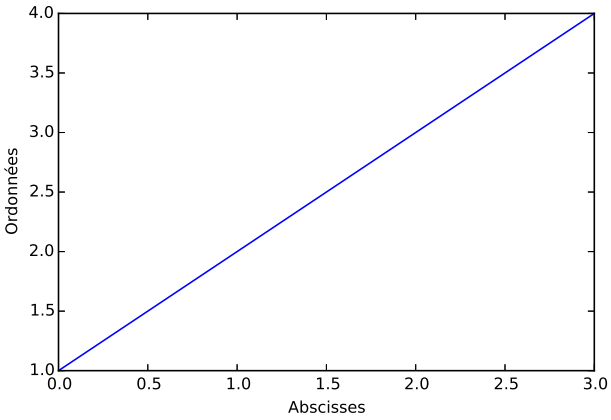


FIGURE 21.1 – mpl-1

Comme on peut le voir, la figure apparaît grâce à `plt.show` placé en fin de programme.

L'instruction `plt.savefig()` permet de sauvegarder la figure.

Une autre possibilité pour obtenir un résultat équivalent est d'utiliser la sous-bibliothèque *pylab*. Les fonctions de *pylab* ressemblent beaucoup aux fonctions graphiques d'autres langages comme *Matlab* ou *Scilab*. Nous privilégierons *pyplot* pour la suite que nous importerons comme suit :

```
import maptplotlib.pyplot as plt
```

Voici un programme un peu plus complet et détaillé :

mpl-3.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  # crée une nouvelle figure 8x6 points, utilisant 100 points par pouce
7  plt.figure(figsize=(8,6), dpi=80)
8
9  # crée une nouvelle sous-fenêtre avec 1 ligne et 1 colonne
10 plt.subplot(111)
11
12 X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
13 C,S = np.cos(X), np.sin(X)
14
15 # cosinus, couleur bleu, ligne continue, taille 1 point
16 plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
17
18 # sinus, de couleur verte
19 plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")
20
21 # limites de x
22 plt.xlim(-4.0,4.0)
23
24 # caractéristiques pour l'axe x
25 plt.xticks(np.linspace(-4,4,9,endpoint=True))
26
27 # limites de y
28 plt.ylim(-1.0,1.0)
29
30 # caractéristiques pour l'axe y
```

```
31  plt.yticks(np.linspace(-1,1,5,endpoint=True))
32
33  # Pour sauvegarder la figure sous forme d'image
34  plt.savefig("mpl-3.eps",dpi=200)
35
36  plt.show()
```

qui donne :

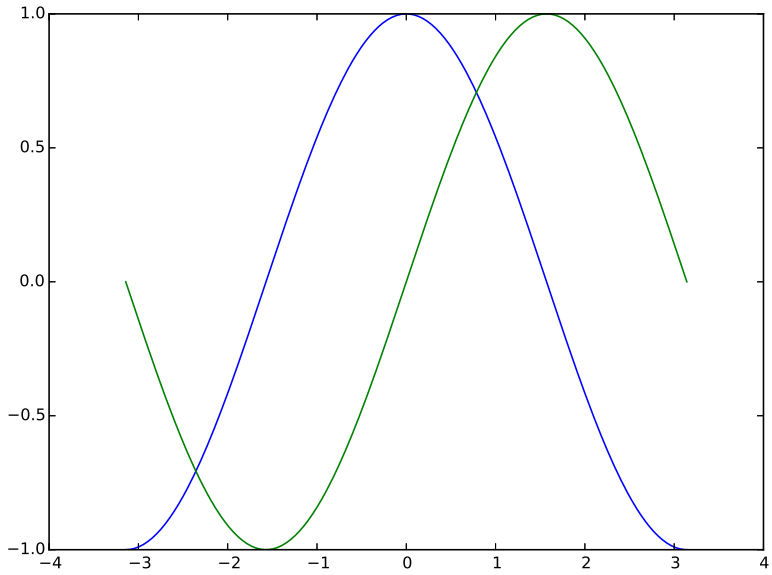


FIGURE 21.2 – mpl-3

On peut modifier la couleur, le style et l'épaisseur des courbes dans l'instruction `plot()`, par exemple, avec :

```
plot(X, C, color="blue", linewidth=2.5,linestyle="--")
```

Quelques options pour `linestyle` :

-	trait plein
–	tirets
-.	alternance tirets - points
:	pointillés

TABLE 21.1 – Options linestyle

On peut modifier les points, avec `marker`. Un exemple ci-dessous, que vous pourrez modifier :

mpl-options

```
1  # -*- coding: utf-8 -*-
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  n = 256
```

```
7  X = np.linspace(-np.pi,np.pi,n,endpoint=True)
8  Y = np.sin(2*X)
9
10 plt.plot (X, Y+1, color='blue', linestyle="-.")
11 plt.plot (X, Y-1, color='red', marker = 'H')
12 plt.savefig("mpl-options.eps",dpi=200)
13 plt.show()
```

qui donne :

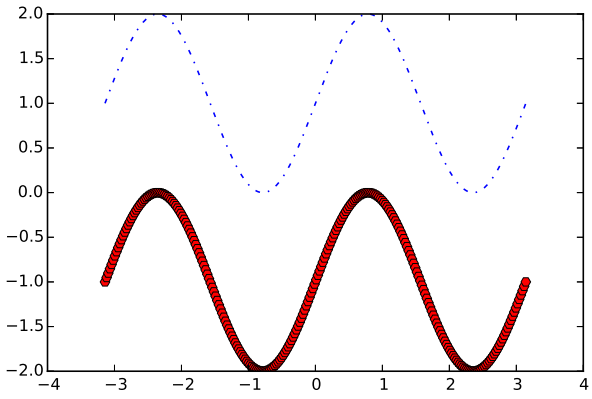


FIGURE 21.3 – mpl-3

Quelques options pour `marker` :

o	cercle	•
D	losange	◊
h	hexagone 1	⬡
H	hexagone 2	⬭
p	pentagone	⬠
^	triangle haut	K
v	triangle bas	L
+	plus	+
s	carré	■

TABLE 21.2 – Options marker

Vous pourrez trouver d'autres options ici par exemple :

http://www.loria.fr/~rougier/teaching/matplotlib/#imshow

On peut aussi modifier les limites ainsi que le pas des abscisses, par exemple pour x :

```
xlim(X.min()*1.1, X.max()*1.1)
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
```

On peut également introduire du code LaTeXpour les différents axes, par exemple dans le programme ci-dessous :

mpl-6.py

```
1  # -*- coding: utf-8 -*-
2
```

```
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  plt.figure(figsize=(8,5), dpi=80)
7  plt.subplot(111)
8
9  X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
10 C,S = np.cos(X), np.sin(X)
11
12 plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
13 plt.plot(X, S, color="red", linewidth=2.5, linestyle="--")
14
15 plt.xlim(X.min()*1.1, X.max()*1.1)
16 plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
17        [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
18
19 plt.ylim(C.min()*1.1,C.max()*1.1)
20 plt.yticks([-1, 0, +1],[r'$-1$', r'$0$', r'$+1$'])
21
22 # Pour sauvegarder la figure sous forme d'image
23 plt.savefig("mpl-6.eps",dpi=200)
24
25 plt.show()
```

On obtient alors :

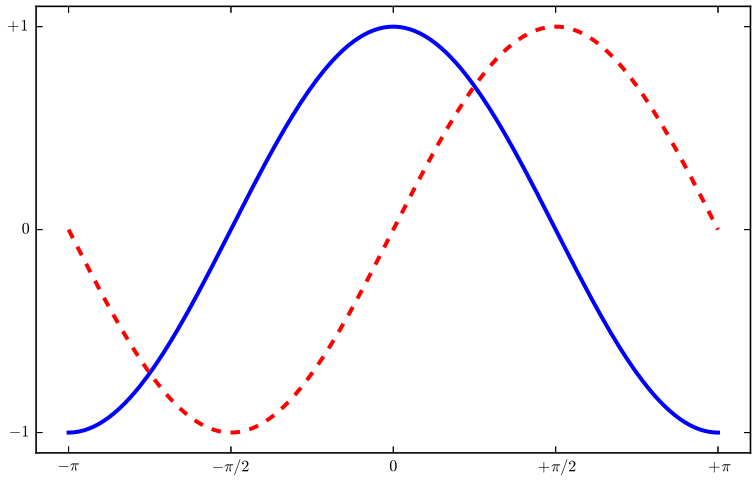


FIGURE 21.4 – mpl-6

On peut également modifier le cadre et la position des axes en insérant par exemple dans le programme (cf. *mpl-7.py*) :

```
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
```

Encore une possibilité : ajouter une légende, à l'aide de l'option `label = " "` dans l'instruction `plot()`, ... :

```
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosinus")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="--",  label="sinus")
```

On obtient ainsi :

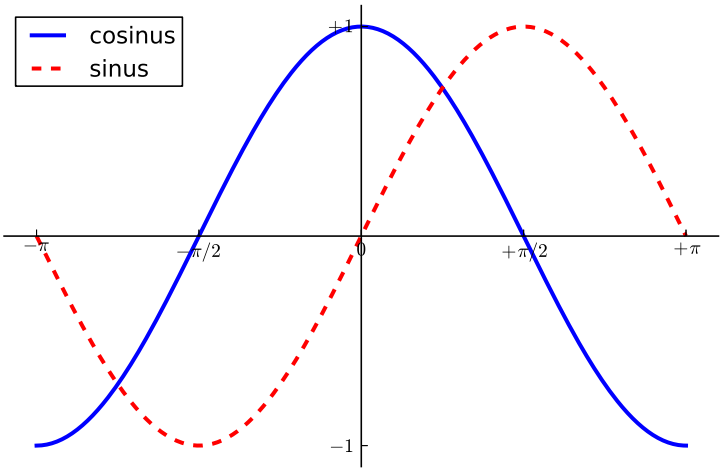


FIGURE 21.5 – mpl-8

Pour faire joli, on peut aussi annoter des points en ajoutant :

```
t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)],
     color ='blue',  linewidth=1.5, linestyle="--")
plt.scatter([t,],[np.cos(t),], 50, color ='blue')
plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$', xy=(t, np.sin(t)),
    xycoords='data',
         xytext=(+10, +30), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)],
          color ='red',  linewidth=1.5, linestyle="--")
plt.scatter([t,],[np.sin(t),], 50, color ='red')
plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$', xy=(t, np.cos(t)),  xycoords='
    data',
         xytext=(-90, -50), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```

On obtient alors :

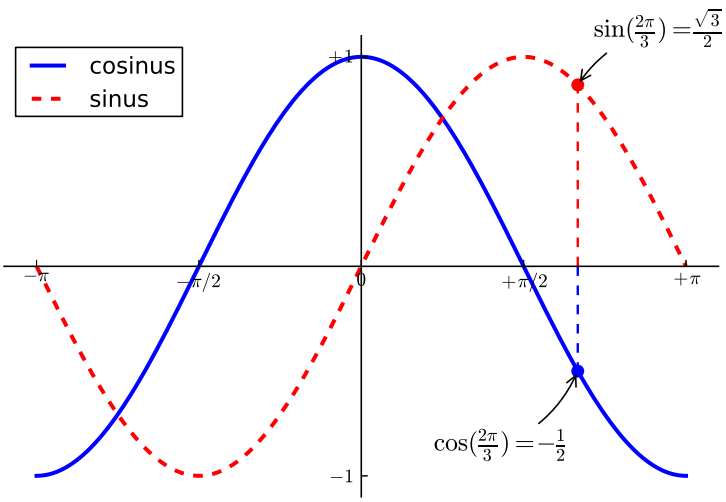


FIGURE 21.6 – mpl-9

Avec une petite amélioration, on obtient le programme suivant :

mpl-10.py

```
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5), dpi=80)
plt.subplot(111)

X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C,S = np.cos(X), np.sin(X)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosinus")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="--",  label="sinus")

ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

plt.xlim(X.min()*1.1, X.max()*1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

plt.ylim(C.min()*1.1,C.max()*1.1)
plt.yticks([-1, +1],
       [r'$-1$', r'$+1$'])

plt.legend(loc='upper left')

t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)],
     color ='blue',  linewidth=.5, linestyle="--")
plt.scatter([t,],[np.cos(t),], 50, color ='blue')
plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$', xy=(t, np.sin(t)),  xycoords='data',
         xytext=(+10, +30), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)],
          color ='red',  linewidth=.5, linestyle="--")
plt.scatter([t,],[np.sin(t),], 50, color ='red')
plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$', xy=(t, np.cos(t)),  xycoords='data',
         xytext=(-90, -50), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(16)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.
```

```
                                                    65 ))
51
52  # Pour sauvegarder la figure sous forme d'image
53  plt.savefig("mpl-10.eps",dpi=200)
54
55  plt.show()
```

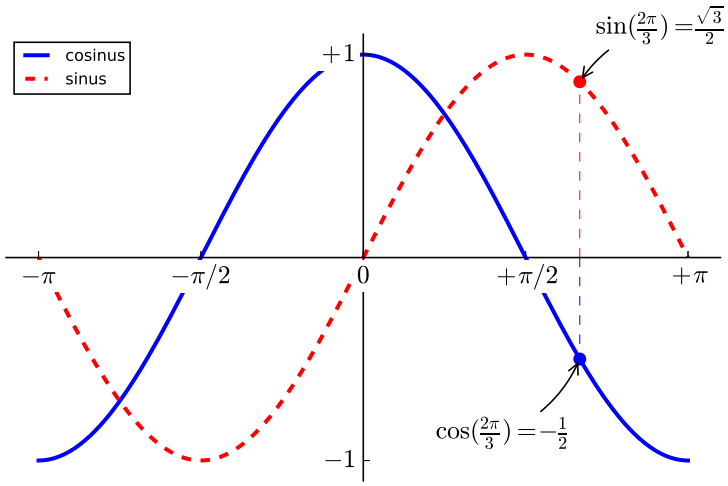


FIGURE 21.7 – mpl-10

21.3 Représentation de données

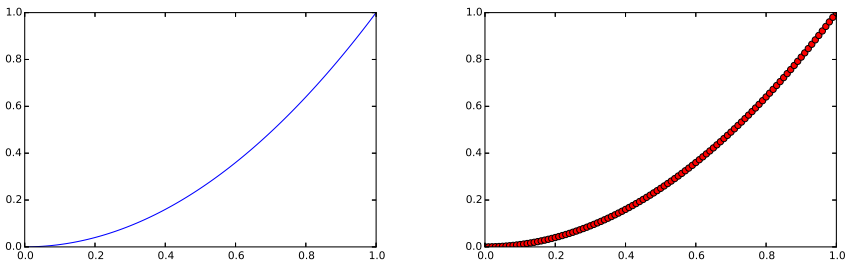
On trouve sur sur la toile des fichiers de données (par exemple des relevés de température) : ce sont par exemple des fichiers ayant l'extension `.dat`. On peut représenter graphiquement ces fichiers. Voici, dans le programme ci-dessous deux méthodes pour obtenir des graphes à partir de ces données :

graphe-donnees-dat.py

```
1   # -*- coding: utf-8 -*-
2
3   # on définit la fonction mathématique
4   def f(x):
5       return(x**2)
6
7   # on ouvre le fichier "donnees" en écriture
8   fichier1 = open('donnees.dat','w')
9   # on remplit le fichier
10  for i in range(101):
11      x = i/100
12      y = f(x)
13      fichier1.write(str(x) + "\t" + str(y) + "\n")
14  # on ferme le fichier
15  fichier1.close()
16  # on peut l'ouvrir ensuite en tant que fichier texte
17
18  # on ouvre le fichier en mode lectire
19  fichier2 = open('donnees.dat','r')
20  # on lit son contenu
21  contenu = fichier2.readlines()
```

```
22  # on ferme le fichier
23  fichier2.close()
24  # on construit les abscisses et ordonnées
25  abscisse = []
26  ordonnee = []
27  for ligne in contenu:
28      elem = ligne.split()
29      abscisse.append(elem[0])
30      ordonnee.append(elem[1])
31
32  # on trace le graphe correspondant
33  import matplotlib.pyplot as plt
34  plt.plot(abscisse,ordonnee)
35  plt.savefig('donnees1.eps',dpi=200)
36  plt.show()
37
38  # une autre façon de procéder :
39  import numpy as np
40  x,y = np.loadtxt("donnees.dat", unpack=True)
41  plt.plot(x,y,'o',color='r')
42  plt.savefig('donnees2.eps',dpi=200)
43  plt.show()
```

qui donne :



(a) donnees1 (b) donnees2

FIGURE 21.8 – Représentation de données `dat`

Il peut parfois être plus pratique de travailler avec des fichiers `.csv` qui sont des fichiers texte utilisables par un tableur (`Excel`, ...). Dans ce cas, on peut utiliser le module *csv* :

graphe-donnees-csv.py

```
1  # -*- coding: utf-8 -*-
2
3  # on définit la fonction mathématique
4  def f(x):
5      return(x**2)
6
7  import csv
8  # on ouvre le fichier "donnees" en écriture
9  with open('donnees.csv','w') as fichier1:
10     entree = csv.writer(fichier1)
11     # on remplit le fichier
12     for i in range(101):
13         x = i/100
14         y = f(x)
15         entree.writerow([x,y])
16 # on peut l'ouvrir ensuite en tant que fichier csv
```

```
17
18  # on initialise les abscisses et ordonnées
19  abscisse, ordonnee = [], []
20  # on ouvre le fichier en mode lectire
21  with open('donnees.csv','r') as fichier2:
22      sortie = csv.reader(fichier2,delimiter=';')
23      # on lit son contenu
24      for ligne in sortie:
25          # on sépare x et y
26          elem=ligne[0].split(',')
27          # on remplit les abscisses et ordonnées
28          abscisse.append(float(elem[0]))
29          ordonnee.append(float(elem[1]))
30
31  # on trace le graphe correspondant
32  import matplotlib.pyplot as plt
33  plt.plot(abscisse,ordonnee)
34  plt.savefig('donnees3.eps',dpi=200)
35  plt.show()
```

On obtient le même graphique.

21.4 Des graphiques plus élaborés

Par ailleurs, il existe de très nombreux exemples de graphiques réalisés avec *MatPlotLib* (codes fournis) sur le site : `matplotlib.org/gallery.html#pylab_examples`.

21.5 Graphiques multiples

Dans le fichier *mpl-multi-graph.py*, il y un exemple de fenêtre graphique multiple :

`subplot(211)` ou `subplot(2, 1, 1)` signifie qu'il y a 2 lignes et que le graphe est sur la colonne 1 et sur la ligne 1.

Programme *mpl-multi-graph.py* :

mpl-multi-graph.py

```
1   # -*- coding: utf-8 -*-
2
3   import numpy as np
4   import matplotlib.pyplot as plt
5
6   # On crée d'abord des tableaux de 100 éléments chacun
7   x1 = np.arange(0.0,10.0,0.1)
8   x2 = np.arange(0.0,5.0,0.05)
9
10  y1 = np.sin(x1)
11  y2 = np.cos(x2)
12
13  # On définit tout d'abord l'existence d'une figure
14  plt.figure(1)
15
16  # Celle-ci compte deux sous-graphiques
17  plt.subplot(211)     # sublot(nbre rangées, nbre colonnes, position)
18  plt.plot(x1,y1,'rs',label='série 1') # Trace des carrés rouges
19  plt.title ('Titre d\'en haut')  # Titre associé à la première figure
20  plt.legend(loc=3)
```

```
21
22  # Et le deuxième
23  plt.subplot(212)    # Deuxième d'une colonne de deux rangées
24  plt.plot(x2,y2,'bo',label='série 2') # Trace des cercles bleus
25  plt.title('Titre d\'en bas')
26
27  # Note: Puisque ces trois commandes apparaissent après la figure 2,
28  # elles ne s'appliquent qu'à celle-ci. Il faudrait répéter pour 1
29  # si on veut les avoir là aussi.
30  plt.xlabel('Abscisse')
31  plt.ylabel('Ordonnée')
32  plt.legend()
33
34  # Pour sauvegarder la figure sous forme d'image
35  plt.savefig("mpl-multi-graph.eps",dpi=200)
36
37  plt.show()
```

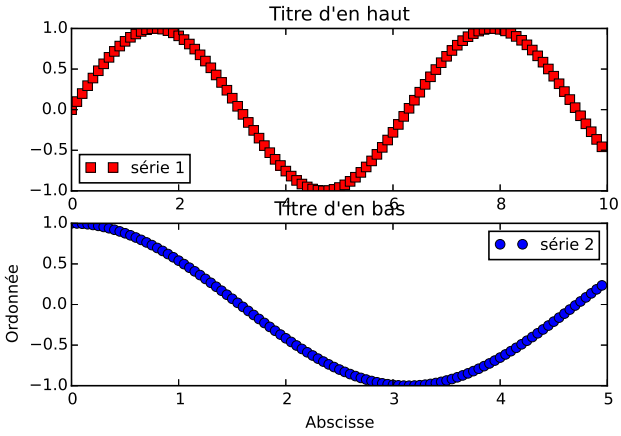


FIGURE 21.9 – mpl-multi-graph

21.6 Graphiques animés

Dans le fichier *mpl-animation.py*, il y un exemple de courbe animée :

mpl-animation.py

```
1  # -*- coding: utf-8 -*-
2
3  """
4  Un exemple simple d'animation
5  """
6
7  import numpy as np
8  import matplotlib.pyplot as plt
9  import matplotlib.animation as animation
10
11 fig = plt.figure()
12 ax = fig.add_subplot(111)
13
14 x = np.arange(0, 2*np.pi, 0.01)
15 line, = ax.plot(x, np.sin(x))
16
```

```
17  def animate(i):
18      line.set_ydata(np.sin(x+i/10.0))
19      return line,
20
21  def init():
22      line.set_ydata(np.ma.array(x, mask=True))
23      return line,
24
25  ani = animation.FuncAnimation(fig, animate, np.arange(1, 200),
                                      init_func=init,
26      interval=25, blit=True)
27  plt.show()
```

Un autre exemple (fichier “mpl-animation2.py”) décrit le mouvement d’un pendule double :

mpl-animation2.py

```
1   # -*- coding: utf-8 -*-
2
3   # Double pendulum formula translated from the C code at
4   # http://www.physics.usyd.edu.au/~wheat/dpend_html/solve_dpend.c
5
6   from numpy import sin, cos, pi, array
7   import numpy as np
8   import matplotlib.pyplot as plt
9   import scipy.integrate as integrate
10  import matplotlib.animation as animation
11
12  G =  9.8 # acceleration due to gravity, in m/s^2
13  L1 = 1.0 # length of pendulum 1 in m
14  L2 = 1.0 # length of pendulum 2 in m
15  M1 = 1.0 # mass of pendulum 1 in kg
16  M2 = 1.0 # mass of pendulum 2 in kg
17
18  def derivs(state, t):
19
20      dydx = np.zeros_like(state)
21      dydx[0] = state[1]
22
23      del_ = state[2]-state[0]
24      den1 = (M1+M2)*L1 - M2*L1*cos(del_)*cos(del_)
25      dydx[1] = (M2*L1*state[1]*state[1]*sin(del_)*cos(del_)
26                 + M2*G*sin(state[2])*cos(del_) + M2*L2*state[3]*state[
                                       3]*sin(del_)
27                 - (M1+M2)*G*sin(state[0]))/den1
28
29      dydx[2] = state[3]
30
31      den2 = (L2/L1)*den1
32      dydx[3] = (-M2*L2*state[3]*state[3]*sin(del_)*cos(del_)
33                 + (M1+M2)*G*sin(state[0])*cos(del_)
34                 - (M1+M2)*L1*state[1]*state[1]*sin(del_)
35                 - (M1+M2)*G*sin(state[2]))/den2
36
37      return dydx
38
39  # create a time array from 0..100 sampled at 0.1 second steps
40  dt = 0.05
41  t = np.arange(0.0, 20, dt)
42
```

```
43  # th1 and th2 are the initial angles (degrees)
44  # w10 and w20 are the initial angular velocities (degrees per second)
45  th1 = 120.0
46  w1 = 0.0
47  th2 = -10.0
48  w2 = 0.0
49
50  rad = pi/180
51
52  # initial state
53  state = np.array([th1, w1, th2, w2])*pi/180.
54
55  # integrate your ODE using scipy.integrate.
56  y = integrate.odeint(derivs, state, t)
57
58  x1 = L1*sin(y[:,0])
59  y1 = -L1*cos(y[:,0])
60
61  x2 = L2*sin(y[:,2]) + x1
62  y2 = -L2*cos(y[:,2]) + y1
63
64  fig = plt.figure()
65  ax = fig.add_subplot(111, autoscale_on=False, xlim=(-2, 2), ylim=(-2,
                                                2))
66  ax.grid()
67
68  line, = ax.plot([], [], 'o-', lw=2)
69  time_template = 'time = %.1fs'
70  time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)
71
72  def init():
73      line.set_data([], [])
74      time_text.set_text('')
75      return line, time_text
76
77  def animate(i):
78      thisx = [0, x1[i], x2[i]]
79      thisy = [0, y1[i], y2[i]]
80
81      line.set_data(thisx, thisy)
82      time_text.set_text(time_template%(i*dt))
83      return line, time_text
84
85  ani = animation.FuncAnimation(fig, animate, np.arange(1, len(y)),
86      interval=25, blit=True, init_func=init)
87
88  # ani.save('mpl_animation2.mp4', fps=15, clear_temp=True)
89  plt.show()
```

21.7 Graphiques en 3 dimensions

Le fichier *mpl-surf.py* est un exemple de représentation en $3D$ d'une fonction à deux variables.

mpl-surf.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
```

```
4  import matplotlib.pyplot as plt
5  from mpl_toolkits.mplot3d import Axes3D
6
7  fig = plt.figure()
8  ax = Axes3D(fig)
9  X = np.arange(-4, 4, 0.25)
10 Y = np.arange(-4, 4, 0.25)
11 X, Y = np.meshgrid(X, Y)
12 R = np.sqrt(X**2 + Y**2)
13 Z = np.sin(R)
14
15 ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)
16 ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=plt.cm.hot)
17 ax.set_zlim(-2,2)
18
19 # Pour sauvegarder la figure sous forme d'image
20 plt.savefig("mpl-surf.eps",dpi=200)
21
22 plt.show()
```

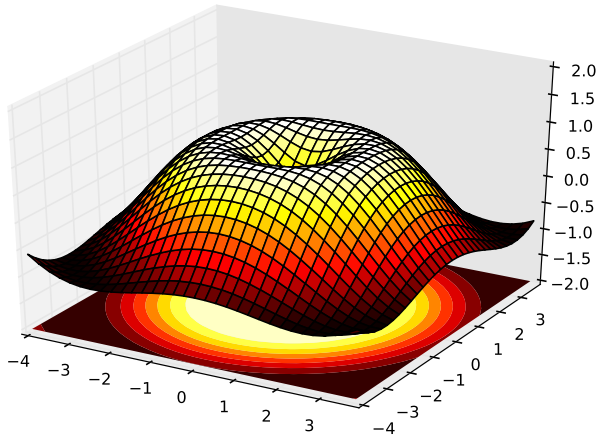


FIGURE 21.10 – mpl-surf

Le fichier *mpl-contour.py* est un exemple de représentation d'une fonction à deux variables en niveaux de couleur et en contours.

mpl-contour.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  def f(x,y):
7      return (1-x/2+x**5+y**3)*np.exp(-x**2-y**2)
8
9  n = 256
10 x = np.linspace(-3,3,n)
11 y = np.linspace(-3,3,n)
12 X,Y = np.meshgrid(x,y)
13
14 plt.axes([0.025,0.025,0.95,0.95])
15
16 plt.contourf(X, Y, f(X,Y), 8, alpha=.75, cmap=plt.cm.hot)
17 C = plt.contour(X, Y, f(X,Y), 8, colors='black', linewidth=.5)
```

```
18  plt.clabel(C, inline=1, fontsize=10)
19
20  plt.xticks([]), plt.yticks([])
21
22  # Pour sauvegarder la figure sous forme d'image
23  plt.savefig("mpl-contour.eps",dpi=200)
24
25  plt.show()
```

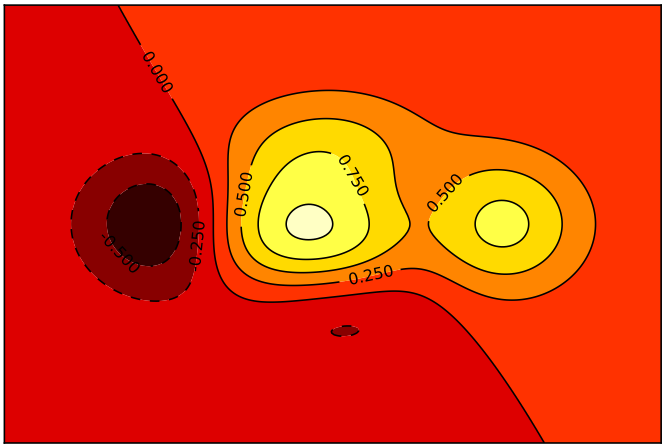


FIGURE 21.11 – mpl-contour

22

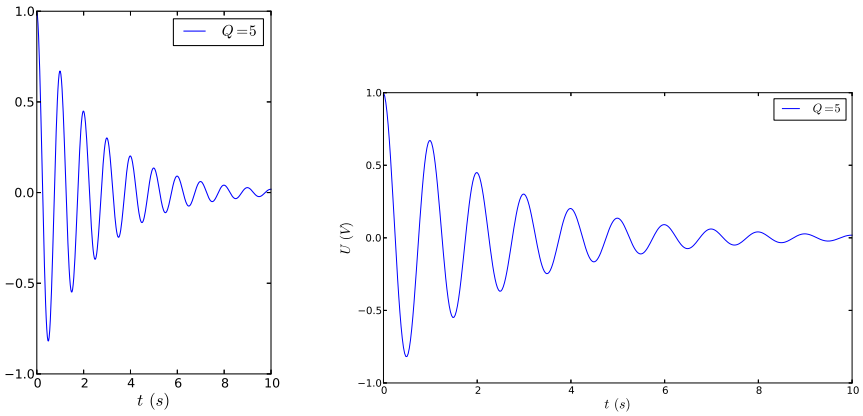
Des graphes en physique - chimie avec Matplotlib

Vous trouverez ici quelques exemples de graphiques élaborés dans le cadre de séances de physique / chimie et qui illustrent les fonctions évoquées dans le chapitre précédent.

filtre1.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Taille de figures
"""

from __future__ import division
from scipy import *
import matplotlib.pyplot as plt

t = linspace(0,10,400)              # Temps = Abscisses

def U(t,Q):
    return exp(-2/Q*t)*cos(2*pi*t)  # Fonction U dépendant d'un param
                                      ètre Q

fig = plt.figure(figsize=(4,6))     # Taille de la figure
plt.plot(t,U(t,5),label="$Q=5$")    # Graphique
plt.xlim(0, 10)                     # Mise en forme
plt.xlabel("$t \, (s)$", fontsize=16)
plt.ylim(-1, 1)
plt.ylabel("$U \, (V)$", fontsize=16)
plt.legend()                        # Appel de la légende

plt.savefig("filtre101.eps",dpi=200)

fig = plt.figure(figsize=(10,6))    # Taille de la figure
plt.plot(t,U(t,5),label="$Q=5$")    # Graphique
plt.xlim(0, 10)                     # Mise en forme
plt.xlabel("$t \, (s)$", fontsize=16)
plt.ylim(-1, 1)
```

```
32  plt.ylabel("$U \, (V)$", fontsize=16)
33  plt.legend()                          # Appel de la légende
34
35  plt.savefig("filtre102.eps",dpi=200)
36  plt.show()
```



(a) filtre101 (b) filtre102

FIGURE 22.1 – Filtres : taille de figures

filtre2.py

```
1   #!/usr/bin/python3
2   # -*- coding: utf-8 -*-
3
4   """
5   Sous graphiques : subplots
6   """
7
8   from __future__ import division
9   from scipy import *
10  import matplotlib.pyplot as plt
11
12  x = logspace(-2,2, 500)  # x=omega/omega0 = Abscisses
13
14  def H(x, Q):       # Fonction de transfert
15    return 1/(1+1j*x/Q-x**2)
16
17  """
18  Rejecteur (1-x**2)/(1+1j*x/Q-x**2)
19  Passe haut -x**2/(1+1j*x/Q-x**2)
20  Passe bas 1/(1+1j*x/Q-x**2)
21  Passe bande 1j*x/Q/(1+1j*x/Q-x**2)
22  """
23
24  Qvaleurs = [0.1, 1/sqrt(2), 3] # Valeurs désirées du facteur de
                                       qualité
25
26  fig = plt.figure()
27  fig.subplots_adjust(hspace=0.1) # Espace entre les graphiques
28
29  # Graphique de l'amplitude
30  ax = fig.add_subplot(211)
```

```
31
32 for Q in Qvaleurs :  # Boucle for pour tracer les graphiques
33   # Légende automatique : conversion en str du facteur de qualité
34   legende="$Q = %4.1f$" %(Q)
35   # Graphiques
36   plt.plot(log10(x), 20*log10(abs(H(x, Q)) ), label=legende,lw=1.5)
37
38 plt.xlim(-2, 2)    # Limites de l'axe des abscisses
39 ax.set_xticklabels([]) # Pas de labels sur l'axe des x
40 plt.ylim(-60., 20)  # Limites de l'axe des ordonnées
41 plt.ylabel("$G^{dB}=20.log_{10}(|H|)$", fontsize=16) # Label de l'axe
                                                        des ordonnées
42
43 plt.legend()    # Appel de la légende
44 plt.grid(True)  # Grille
45
46 # Graphique de la phase
47 ax2 = fig.add_subplot(212)
48
49 for Q in Qvaleurs :
50   legende="$Q = %4.0f$" %(Q)
51   plt.plot(log10(x), 180/3.1416*angle(H(x, Q)), label=legende,lw=1.5)
52
53 plt.xlim(-2, 2)
54 plt.xlabel("$log_{10}(x) = log_{10}(\omega/\omega_{0})$", fontsize=16
                                                            )
55 plt.ylim(-180., 0)
56 plt.ylabel("$\phi \, (^\circ)$", fontsize=16)
57
58 plt.grid(True)
59
60 plt.savefig("filtre2.eps",dpi=200)
61 plt.show()
```

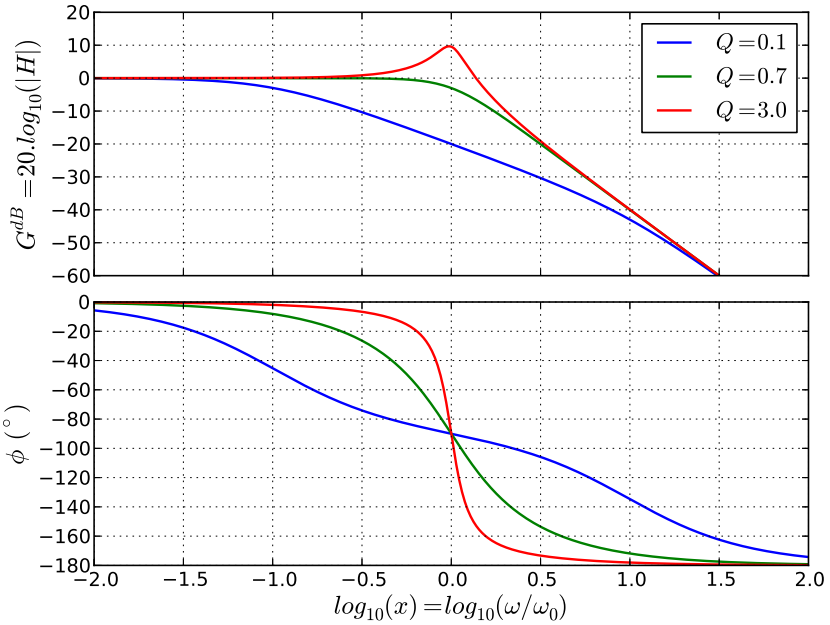


FIGURE 22.2 – Filtre 2

filtre3.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Graphiques en coordonnées semilogarithmiques ou logarithmiques
"""

from __future__ import division
from scipy import *
import matplotlib.pyplot as plt

xlin = linspace(0,2,400)  # Abscisse linéaire
xlog = logspace(-2,4,400) # Abscisse logarithmique

def a(x, Q):  # Module d'une fonction de transfert
  return abs(1/(1+1j*x/Q-x**2))

# axes linéaires
plt.subplot(221)
plt.plot(xlin, a(xlin, 5))
plt.title("linéaire")
plt.grid(True)

# axe y logarithmique
plt.subplot(222)
plt.semilogy(xlin, a(xlin, 5))
plt.title("semi-log y")
plt.grid(True)

# axe x logarithmique
plt.subplot(223)
plt.semilogx(xlog, a(xlin, 5))
plt.title("semi-log x")
plt.grid(True)

# axes logarithmiques
plt.subplot(224)
plt.loglog(xlog, a(xlin, 5), basex=10)
plt.grid(True)
plt.title("log-log base 10 sur x")

plt.suptitle("Systèmes de coordonnées",fontsize=16)
plt.savefig("filtre3.eps",dpi=200)
plt.show()
```

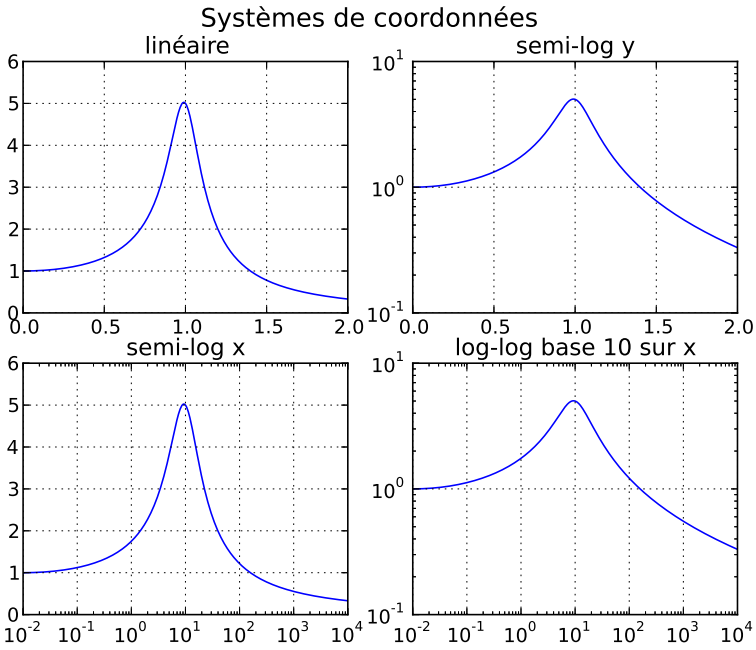


FIGURE 22.3 – Filtre 3

filtre4.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Création d'une légende automatique en fonction d'un paramètre
"""

from __future__ import division
from scipy import *
import matplotlib.pyplot as plt

x = linspace(0,2,400) # x=omega/omega0 = Abscisses

def A(x,Q): # Fonction dépendant d'un paramètre Q
    return 1/sqrt((1-x**2)**2+1/Q**2*x**2)

Qvaleurs = [0.1, 1/sqrt(2), 3, 5]  # Valeurs désirées du facteur de qualité

for Q in Qvaleurs :   # Boucle for pour tracer les graphiques
         # Légende automatique : conversion en str du facteur de qualité
         legende="$Q = %4.1f$" %(Q)
         plt.plot(x, A(x, Q), label=legende,lw=1.5)  # Graphique

plt.xlim(0, 2)  # Limites de l'axe des abscisses
plt.xlabel("$x=\omega/\omega_{0} \,$", fontsize=16)

plt.ylim(0, 5.1)  # Limites de l'axe des ordonnées
plt.ylabel("$A/A_{0} \, $", fontsize=16)
plt.legend()

plt.savefig("filtre4.eps",dpi=200)
```

```
32  plt.show()
```

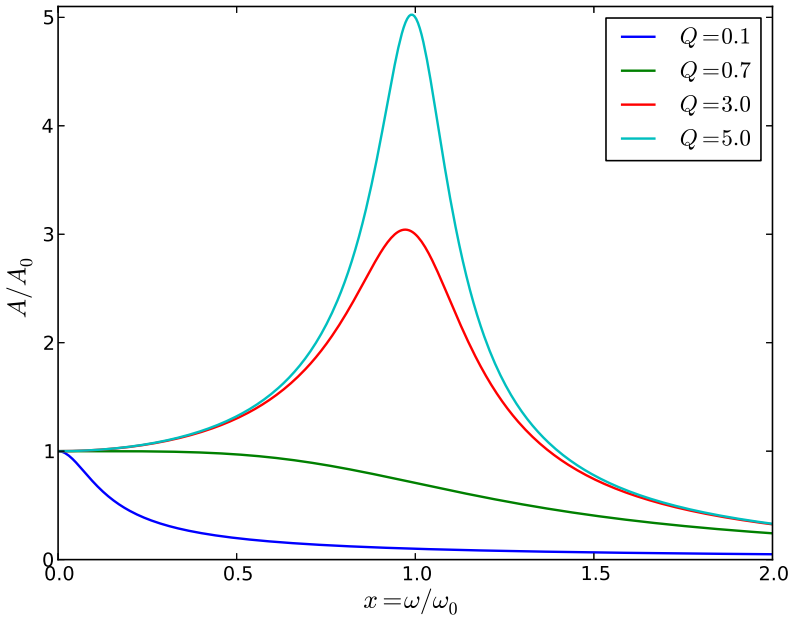


FIGURE 22.4 – Filtre 4

filtre5.py

```
1   #!/usr/bin/python3
2   # -*- coding: utf-8 -*-
3
4   """
5   Graphique avec deux échelles en Y
6   """
7
8   from __future__ import division
9   from scipy import *
10  import matplotlib.pyplot as plt
11
12  x = linspace(0,2,400)   # x=omega/omega0 = Abscisses
13
14  def H(x, Q):      # Fonction de transfert
15    return 1/(1+1j*x/Q-x**2)
16
17  fig = plt.figure()
18  ax1 = fig.add_subplot(111)
19
20  # Graphique 1 avec le système d'axes 1
21  ax1.plot(x, abs(H(x, 5)), '-b', label="$A/A_{0}$",lw=1.5)
22
23  plt.ylim(0, 5.1)   # Limites de l'axe des ordonnées 1
24  plt.ylabel("$A/A_{0} \, $", color='b', fontsize=16)
25  for tl in ax1.get_yticklabels(): # Couleur des étiquettes de l'axe
                                     des ordonnées 1
26      tl.set_color('b')
27
28  plt.legend(loc=2)    # Appel de la légende 1
```

```
29
30 ax2 = ax1.twinx()    # Système d'axes 2
31
32 # Graphique 2 avec le système d'axes 2
33 ax2.plot(x, 180/3.1416*angle(H(x, 5)),'--r', label="$\phi \, (^\circ)
                                        $",lw=1.5)
34
35 plt.ylim(-180, 0)    # Limites de l'axe des ordonnées 2
36 plt.ylabel("$\phi \, (^\circ)$", color='r', fontsize=16)
37 for tl in ax2.get_yticklabels():  # Couleur des étiquettes de l'axe
                                        des ordonnées 2
38     tl.set_color('r')
39
40 plt.legend(loc=1)     # Appel de la légende 2
41 plt.savefig("filtre5.eps",dpi=200)
42
43 plt.show()
```

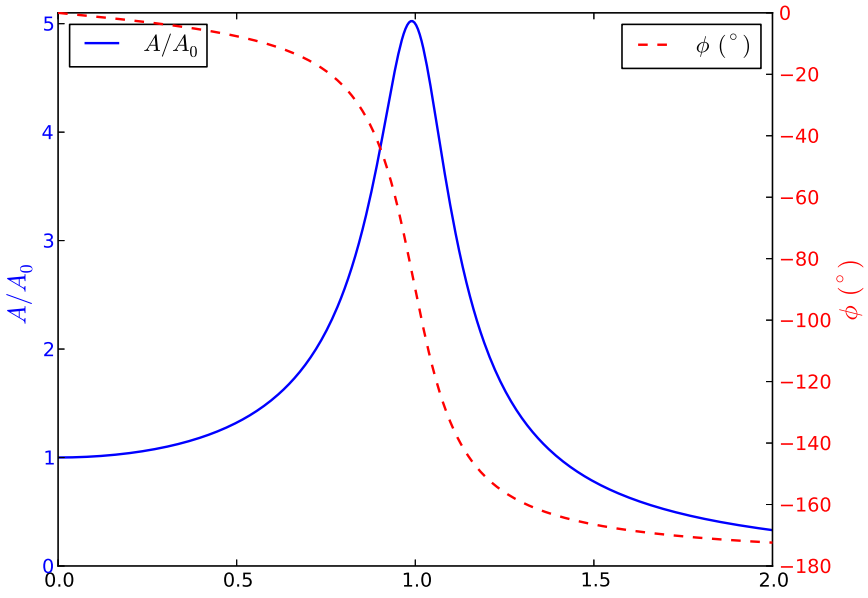


FIGURE 22.5 – Filtre 5

23

Autres exemples de graphes avec Matplotlib

Vous trouverez dans ce chapitre des exemples de graphiques plus complets mais que nous n'étudierons pas.

fromage.py

```
#-*- coding: utf-8 -*-

import matplotlib.pyplot as plt

matieres = 'Maths', 'Physique', 'Chimie', 'Informatique', 'Anglais',
                                        'Français', 'Techno', 'Sport'
parts = [5, 10, 8, 25, 12, 9, 20, 11]
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', '
                                        purple', 'grey', 'orange', '
                                        green']
sortie = (0, 0.2, 0, 0.05, 0, 0, 0,0.1)

plt.pie(parts, explode=sortie, labels=matieres, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.savefig('camembert.eps')
plt.show()
```

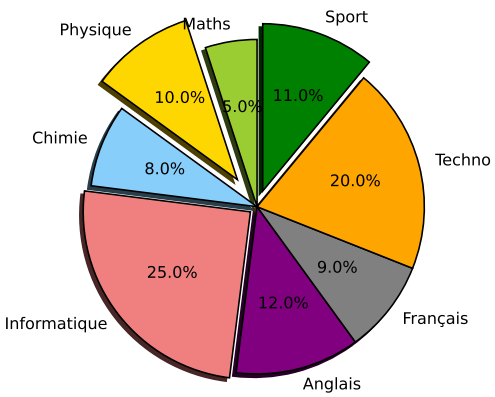


FIGURE 23.1 – camembert

mpl-3ddemo-3.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  fig = plt.figure()
8  ax = fig.gca(projection='3d')
9
10 x = np.linspace(0, 1, 100)
11 y = np.sin(x * 2 * np.pi) / 2 + 0.5
12 ax.plot(x, y, zs=0, zdir='z', label='zs=0, zdir=z')
13
14 colors = ('r', 'g', 'b', 'k')
15 for c in colors:
16     x = np.random.sample(20)
17     y = np.random.sample(20)
18     ax.scatter(x, y, 0, zdir='y', c=c)
19
20 ax.legend()
21 ax.set_xlim3d(0, 1)
22 ax.set_ylim3d(0, 1)
23 ax.set_zlim3d(0, 1)
24
25 # Pour sauvegarder la figure sous forme d'image
26 plt.savefig("mpl-3ddemo-3.eps",dpi=200)
27
28 plt.show()
```

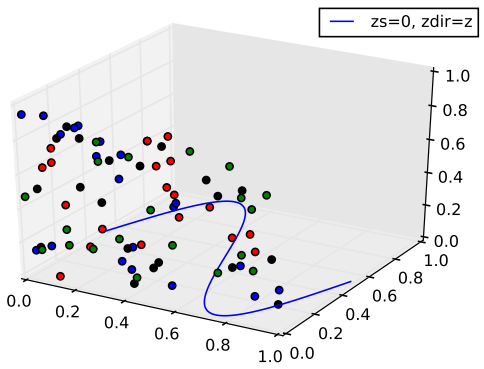


FIGURE 23.2 – mpl-3ddemo-3

mpl-annotation-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from pylab import *
4  from matplotlib.pyplot import figure, show
5  from matplotlib.patches import Ellipse
```

```
6  import numpy as np
7
8  if 1:
9      fig = figure()
10     ax = fig.add_subplot(111, autoscale_on=False, xlim=(-1,5), ylim=(
                                    -3,5))
11
12     t = np.arange(0.0, 5.0, 0.01)
13     s = np.cos(2*np.pi*t)
14     line, = ax.plot(t, s, lw=3, color='purple')
15
16     ax.annotate('axes center', xy=(.5, .5),  xycoords='axes fraction'
                                    ,
17                 horizontalalignment='center', verticalalignment='
                                    center')
18
19     ax.annotate('pixels', xy=(20, 20),  xycoords='figure pixels')
20
21     ax.annotate('points', xy=(100, 300),  xycoords='figure points')
22
23     ax.annotate('offset', xy=(1, 1),  xycoords='data',
24                 xytext=(-15, 10), textcoords='offset points',
25                 arrowprops=dict(facecolor='black', shrink=0.05),
26                 horizontalalignment='right', verticalalignment='
                                    bottom',
27                 )
28
29     ax.annotate('local max', xy=(3, 1),  xycoords='data',
30                 xytext=(0.8, 0.95), textcoords='axes fraction',
31                 arrowprops=dict(facecolor='black', shrink=0.05),
32                 horizontalalignment='right', verticalalignment='top',
33                 )
34
35     ax.annotate('a fractional title', xy=(.025, .975),
36                 xycoords='figure fraction',
37                 horizontalalignment='left', verticalalignment='top',
38                 fontsize=20)
39
40     ax.annotate('bottom right (points)', xy=(-10, 10),
41                 xycoords='axes points',
42                 horizontalalignment='right', verticalalignment='
                                    bottom',
43                 fontsize=20)
44 plt.savefig("mpl-annotation-demo1.eps",dpi=200)
45 show()
46
47
48 if 1:
49     fig = figure()
50     ax = fig.add_subplot(111, polar=True)
51     r = np.arange(0,1,0.001)
52     theta = 2*2*np.pi*r
53     line, = ax.plot(theta, r, color='#ee8d18', lw=3)
54
55     ind = 800
56     thisr, thistheta = r[ind], theta[ind]
57     ax.plot([thistheta], [thisr], 'o')
58     ax.annotate('a polar annotation',
59                 xy=(thistheta, thisr),  # theta, radius
60                 xytext=(0.05, 0.05),    # fraction, fraction
```

```
61                  textcoords='figure fraction',
62                  arrowprops=dict(facecolor='black', shrink=0.05),
63                  horizontalalignment='left',
64                  verticalalignment='bottom',
65                  )
66
67  plt.savefig("mpl-annotation-demo2.eps",dpi=200)
68
69  if 1:
70      el = Ellipse((0,0), 10, 20, facecolor='r', alpha=0.5)
71
72      fig = figure()
73      ax = fig.add_subplot(111, aspect='equal')
74      ax.add_artist(el)
75      el.set_clip_box(ax.bbox)
76      ax.annotate('the top',
77                  xy=(np.pi/2., 10.),      # theta, radius
78                  xytext=(np.pi/3, 20.),   # theta, radius
79                  xycoords='polar',
80                  textcoords='polar',
81                  arrowprops=dict(facecolor='black', shrink=0.05),
82                  horizontalalignment='left',
83                  verticalalignment='bottom',
84                  clip_on=True, # clip to the axes bounding box
85       )
86
87      ax.set_xlim(-20, 20)
88      ax.set_ylim(-20, 20)
89
90  savefig("mpl-annotation-demo3.eps",dpi=200)
91
92  show()
```

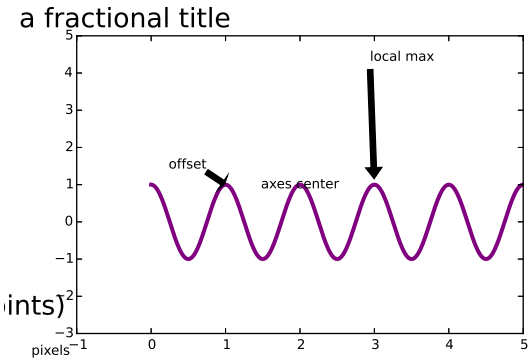


FIGURE 23.3 – mpl-annotation-demo1

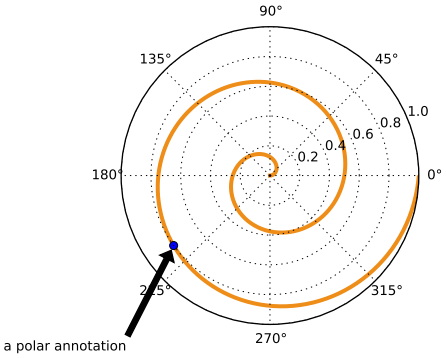


FIGURE 23.4 – mpl-annotation-demo2

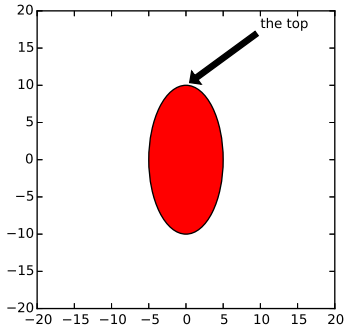


FIGURE 23.5 – mpl-annotation-demo3

mpl-contour3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import axes3d
4  import matplotlib.pyplot as plt
5
6  fig = plt.figure()
7  ax = fig.add_subplot(111, projection='3d')
8  X, Y, Z = axes3d.get_test_data(0.05)
9  cset = ax.contour(X, Y, Z, cmap=plt.cm.coolwarm)
10 # on peut mettre seulement cmap = cm.coolwarm
11 # mais il faut importer le sous-module cm
12 ax.clabel(cset, fontsize=9, inline=1)
13
14 plt.savefig("mpl-contour3d-demo.eps",dpi=200)
15 plt.show()
```

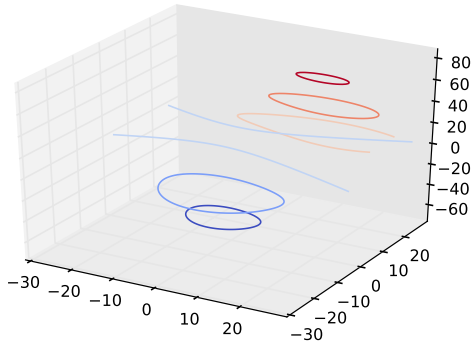


FIGURE 23.6 – mpl-contour3d-demo

mpl-contour3d-demo2.py

```
1  # -*- coding: utf-8 -*-
```

```
2
3  from mpl_toolkits.mplot3d import axes3d
4  import matplotlib.pyplot as plt
5
6  fig = plt.figure()
7  ax = fig.gca(projection='3d')
8  X, Y, Z = axes3d.get_test_data(0.05)
9  cset = ax.contour(X, Y, Z, extend3d=True, cmap=plt.cm.coolwarm)
10 # on peut mettre seulement cmap = cm.coolwarm
11 # mais il faut importer le sous-module cm
12 ax.clabel(cset, fontsize=9, inline=1)
13
14 plt.savefig("mpl-contour3d-demo2.eps",dpi=200)
15 plt.show()
```

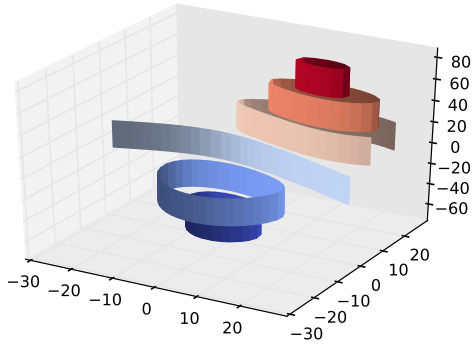


FIGURE 23.7 – mpl-contour3d-demo2

mpl-contour3d-demo3.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import axes3d
4  import matplotlib.pyplot as plt
5
6  fig = plt.figure()
7  ax = fig.gca(projection='3d')
8  X, Y, Z = axes3d.get_test_data(0.05)
9  ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)
10 cset = ax.contour(X, Y, Z, zdir='z', offset=-100, cmap=plt.cm.
                                          coolwarm)
11 cset = ax.contour(X, Y, Z, zdir='x', offset=-40, cmap=plt.cm.coolwarm
                                          )
12 cset = ax.contour(X, Y, Z, zdir='y', offset=40, cmap=plt.cm.coolwarm)
13 # on peut mettre seulement cmap = cm.coolwarm
14 # mais il faut importer le sous-module cm
15
16 ax.set_xlabel('X')
17 ax.set_xlim(-40, 40)
18 ax.set_ylabel('Y')
19 ax.set_ylim(-40, 40)
20 ax.set_zlabel('Z')
21 ax.set_zlim(-100, 100)
```

```
22
23 plt.savefig("mpl-contour3d-demo3.eps",dpi=200)
24 plt.show()
```

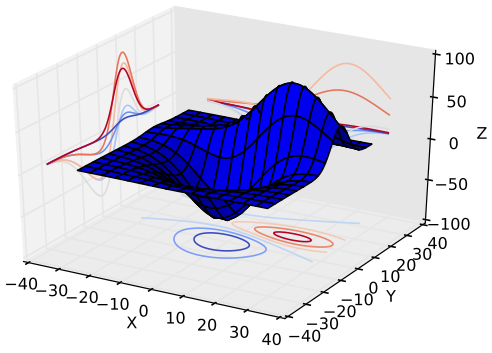


FIGURE 23.8 – mpl-contour3d-demo3

mpl-contourf3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import axes3d
4  import matplotlib.pyplot as plt
5
6  fig = plt.figure()
7  ax = fig.add_subplot(111, projection='3d')
8  X, Y, Z = axes3d.get_test_data(0.05)
9  cset = ax.contour(X, Y, Z, cmap=plt.cm.coolwarm)
10 # on peut mettre seulement cmap = cm.coolwarm
11 # mais il faut importer le sous-module cm
12 ax.clabel(cset, fontsize=9, inline=1)
13
14
15 plt.savefig("mpl-contourf3d-demo.eps",dpi=200)
16 plt.show()
```

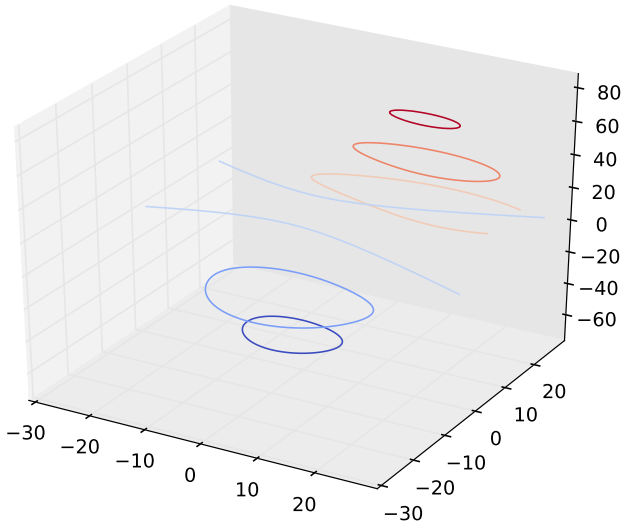


FIGURE 23.9 – mpl-contourf3d-demo

mpl-histogramme3d.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  import matplotlib.pyplot as plt
5  import numpy as np
6
7  fig = plt.figure()
8  ax = fig.add_subplot(111, projection='3d')
9  for c, z in zip(['r', 'g', 'b', 'y'], [30, 20, 10, 0]):
10     xs = np.arange(20)
11     ys = np.random.rand(20)
12
13     cs = [c] * len(xs)
14     cs[0] = 'c'
15     ax.bar(xs, ys, zs=z, zdir='y', color=cs, alpha=0.8)
16
17 ax.set_xlabel('X')
18 ax.set_ylabel('Y')
19 ax.set_zlabel('Z')
20
21 plt.savefig("mpl-histogramme3d.eps",dpi=200)
22 plt.show()
```

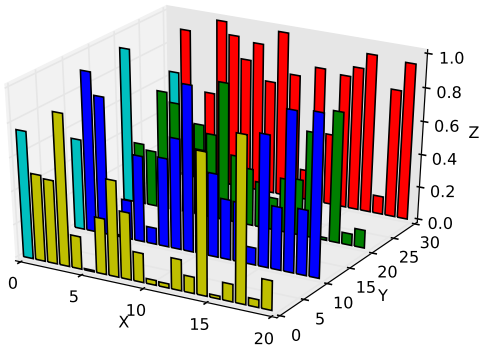


FIGURE 23.10 – mpl-histogramme3d

mpl-parametric1.py

```
1  # -*- coding: utf-8 -*-
2
3  import matplotlib as mpl
4  from mpl_toolkits.mplot3d import Axes3D
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  mpl.rcParams['legend.fontsize'] = 10
9
10 fig = plt.figure()
11 ax = fig.gca(projection='3d')
12 theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
13 z = np.linspace(-2, 2, 100)
14 r = z**2 + 1
15 x = r * np.sin(theta)
16 y = r * np.cos(theta)
17 ax.plot(x, y, z, label='courbe paramétrique')
18 ax.legend()
19
20 plt.savefig("mpl-parametric1.eps",dpi=200)
21 plt.show()
```

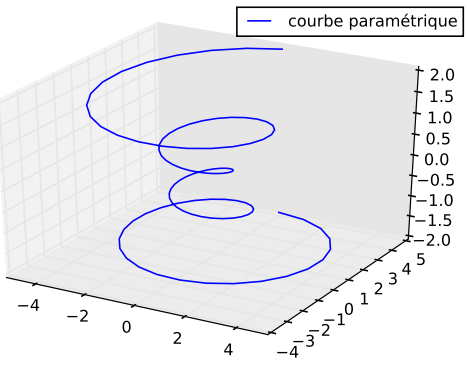


FIGURE 23.11 – mpl-parametric1

mpl-points3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  from mpl_toolkits.mplot3d import Axes3D
5  import matplotlib.pyplot as plt
6
7  def randrange(n, vmin, vmax):
8      return (vmax-vmin)*np.random.rand(n) + vmin
9
10 fig = plt.figure()
11 ax = fig.add_subplot(111, projection='3d')
12 n = 100
13 for c, m, zl, zh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
14     xs = randrange(n, 23, 32)
15     ys = randrange(n, 0, 100)
16     zs = randrange(n, zl, zh)
17     ax.scatter(xs, ys, zs, c=c, marker=m)
18
19 ax.set_xlabel('X Label')
20 ax.set_ylabel('Y Label')
21 ax.set_zlabel('Z Label')
22
23 plt.savefig("mpl-points3d-demo.eps",dpi=200)
24 plt.show()
```

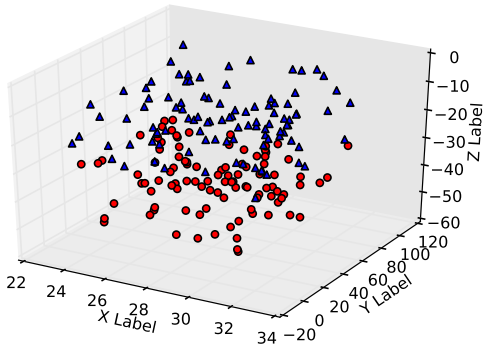


FIGURE 23.12 – mpl-points3d-demo

mpl-poly3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  from matplotlib.collections import PolyCollection
5  from matplotlib.colors import colorConverter
6  import matplotlib.pyplot as plt
7  import numpy as np
8
9  fig = plt.figure()
10 ax = fig.gca(projection='3d')
11
```

```
12  cc = lambda arg: colorConverter.to_rgba(arg, alpha=0.6)
13
14  xs = np.arange(0, 10, 0.4)
15  verts = []
16  zs = [0.0, 1.0, 2.0, 3.0]
17  for z in zs:
18      ys = np.random.rand(len(xs))
19      ys[0], ys[-1] = 0, 0
20      verts.append(list(zip(xs, ys)))
21
22  poly = PolyCollection(verts, facecolors = [cc('r'), cc('g'), cc('b'),
23                                             cc('y')])
24  poly.set_alpha(0.7)
25  ax.add_collection3d(poly, zs=zs, zdir='y')
26
27  ax.set_xlabel('X')
28  ax.set_xlim3d(0, 10)
29  ax.set_ylabel('Y')
30  ax.set_ylim3d(-1, 4)
31  ax.set_zlabel('Z')
32  ax.set_zlim3d(0, 1)
33
34  plt.savefig("mpl-poly3d-demo.eps",dpi=200)
35  plt.show()
```

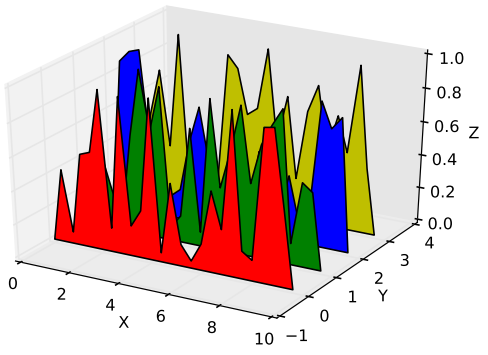


FIGURE 23.13 – mpl-poly3d-demo

mpl-subplot1.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  import matplotlib.pyplot as plt
5  import numpy as np
6
7  def f(t):
8      s1 = np.cos(2*np.pi*t)
9      e1 = np.exp(-t)
10     return np.multiply(s1,e1)
11
12 # première sous-figure
13 t1 = np.arange(0.0, 5.0, 0.1)
```

```
14  t2 = np.arange(0.0, 5.0, 0.02)
15  t3 = np.arange(0.0, 2.0, 0.01)
16
17  fig = plt.figure(figsize=plt.figaspect(2.))
18  fig.suptitle('A tale of 2 subplots')
19  ax = fig.add_subplot(2, 1, 1)
20  l = ax.plot(t1, f(t1), 'bo',
21              t2, f(t2), 'k--', markerfacecolor='green')
22  ax.grid(True)
23  ax.set_ylabel('Damped oscillation')
24
25  # deuxième sous-figure
26  ax = fig.add_subplot(2, 1, 2, projection='3d')
27  X = np.arange(-5, 5, 0.25)
28  xlen = len(X)
29  Y = np.arange(-5, 5, 0.25)
30  ylen = len(Y)
31  X, Y = np.meshgrid(X, Y)
32  R = np.sqrt(X**2 + Y**2)
33  Z = np.sin(R)
34
35  surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
36          linewidth=0, antialiased=False)
37
38  ax.set_zlim3d(-1, 1)
39
40  plt.savefig("mpl-subplot1.eps",dpi=200)
41  plt.show()
```

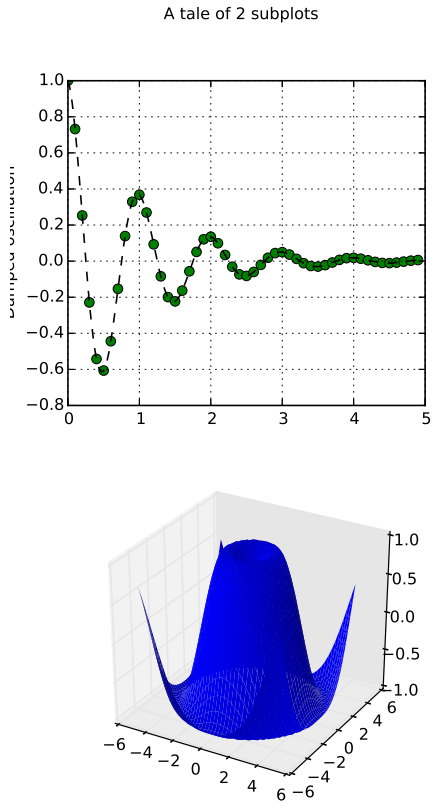


FIGURE 23.14 – mpl-subplot1

mpl-subplot3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d.axes3d import Axes3D
4  import matplotlib.pyplot as plt
5
6  import numpy as np
7  from mpl_toolkits.mplot3d.axes3d import get_test_data
8
9  fig = plt.figure(figsize=plt.figaspect(0.5))
10
11 # première sous-figure
12 ax = fig.add_subplot(1, 2, 1, projection='3d')
13 X = np.arange(-5, 5, 0.25)
14 Y = np.arange(-5, 5, 0.25)
15 X, Y = np.meshgrid(X, Y)
16 R = np.sqrt(X**2 + Y**2)
17 Z = np.sin(R)
18 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.
                                                    coolwarm,
19         linewidth=0, antialiased=False)
20 ax.set_zlim3d(-1.01, 1.01)
21
22 fig.colorbar(surf, shrink=0.5, aspect=10)
23
24 # deuxième sous-figure
25 ax = fig.add_subplot(1, 2, 2, projection='3d')
26 X, Y, Z = get_test_data(0.05)
27 ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
28
29 plt.savefig("mpl-subplot3d-demo.eps",dpi=200)
30
31 plt.show()
```

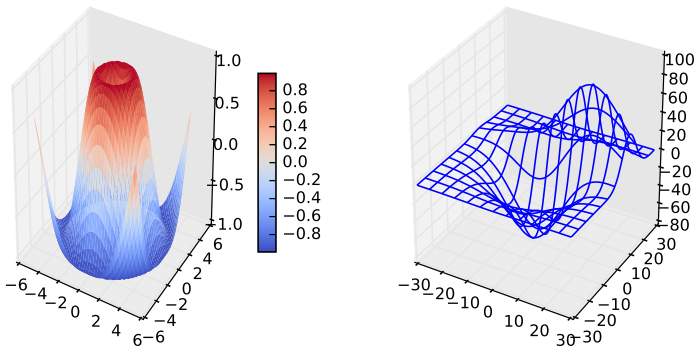


FIGURE 23.15 – mpl-subplot3d-demo

mpl-surface3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  from matplotlib import cm
5  from matplotlib.ticker import LinearLocator, FormatStrFormatter
6  import matplotlib.pyplot as plt
7  import numpy as np
8
9  fig = plt.figure()
10 ax = fig.gca(projection='3d')
11 X = np.arange(-5, 5, 0.25)
12 Y = np.arange(-5, 5, 0.25)
13 X, Y = np.meshgrid(X, Y)
14 R = np.sqrt(X**2 + Y**2)
15 Z = np.sin(R)
16 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.
                                                    coolwarm,
17         linewidth=0, antialiased=False)
18 ax.set_zlim(-1.01, 1.01)
19
20 ax.zaxis.set_major_locator(LinearLocator(10))
21 ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
22
23 fig.colorbar(surf, shrink=0.5, aspect=5)
24
25 plt.savefig("mpl-surface3d-demo.eps",dpi=200)
26
27 plt.show()
```

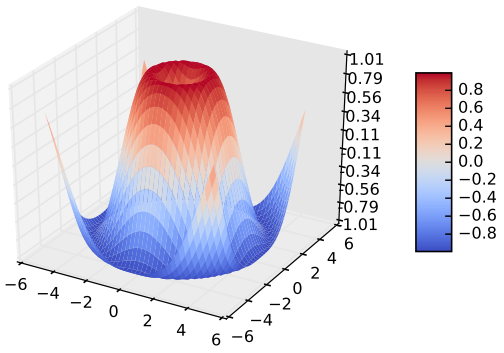


FIGURE 23.16 – mpl-surface3d-demo

mpl-surface3d-demo3.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  from matplotlib import cm
5  from matplotlib.ticker import LinearLocator
6  import matplotlib.pyplot as plt
7  import numpy as np
```

```
8
9  fig = plt.figure()
10 ax = fig.gca(projection='3d')
11 X = np.arange(-5, 5, 0.25)
12 xlen = len(X)
13 Y = np.arange(-5, 5, 0.25)
14 ylen = len(Y)
15 X, Y = np.meshgrid(X, Y)
16 R = np.sqrt(X**2 + Y**2)
17 Z = np.sin(R)
18
19 colortuple = ('y', 'b')
20 colors = np.empty(X.shape, dtype=str)
21 for y in range(ylen):
22     for x in range(xlen):
23         colors[x, y] = colortuple[(x + y) % len(colortuple)]
24
25 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, facecolors=
                                        colors,
26         linewidth=0, antialiased=False)
27
28 ax.set_zlim3d(-1, 1)
29 ax.w_zaxis.set_major_locator(LinearLocator(6))
30
31 plt.savefig("mpl-surface3d-demo3.eps",dpi=200)
32 plt.show()
```

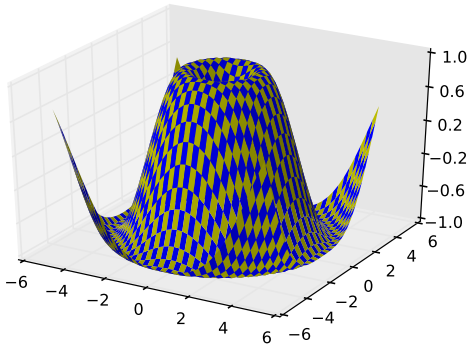


FIGURE 23.17 – mpl-surface3d-demo3

mpl-surf-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import axes3d
4  import matplotlib.pyplot as plt
5  import numpy as np
6
7  fig = plt.figure()
8  ax = fig.add_subplot(111, projection='3d')
9  X, Y, Z = axes3d.get_test_data(0.05)
10 ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
11
```

```
12  plt.savefig("mpl-surf-demo.eps",dpi=200)
13  plt.show()
```

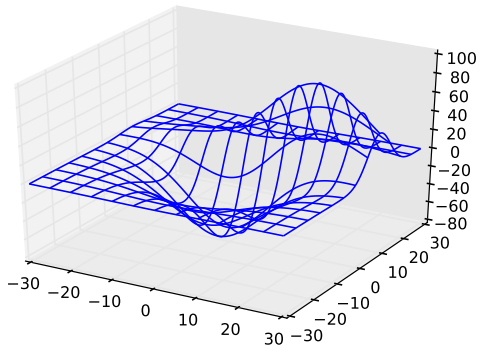


FIGURE 23.18 – mpl-surf-demo

mpl-surf-demo2.py

```
1   # -*- coding: utf-8 -*-
2
3   from mpl_toolkits.mplot3d import Axes3D
4   from matplotlib import cm
5   import matplotlib.pyplot as plt
6   import numpy as np
7
8   n_angles = 36
9   n_radii = 8
10
11  radii = np.linspace(0.125, 1.0, n_radii)
12
13  angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)
14
15  angles = np.repeat(angles[...,np.newaxis], n_radii, axis=1)
16
17  x = np.append(0, (radii*np.cos(angles)).flatten())
18  y = np.append(0, (radii*np.sin(angles)).flatten())
19
20  z = np.sin(-x*y)
21
22  fig = plt.figure()
23  ax = fig.gca(projection='3d')
24
25  ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
26
27  plt.savefig("mpl-surf-demo2.eps",dpi=200)
28  plt.show()
```

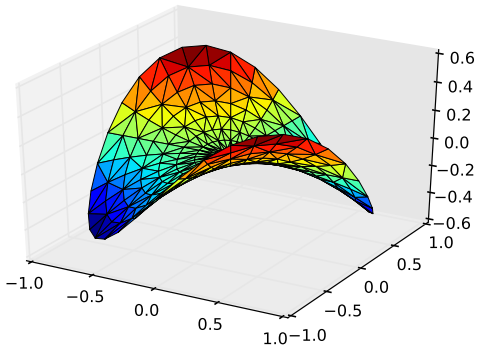


FIGURE 23.19 – mpl-surf-demo2

mpl-text3d-demo.py

```
1  # -*- coding: utf-8 -*-
2
3  from mpl_toolkits.mplot3d import Axes3D
4  import matplotlib.pyplot as plt
5
6  fig = plt.figure()
7  ax = fig.gca(projection='3d')
8
9  zdirs = (None, 'x', 'y', 'z', (1, 1, 0), (1, 1, 1))
10 xs = (2, 6, 4, 9, 7, 2)
11 ys = (6, 4, 8, 7, 2, 2)
12 zs = (4, 2, 5, 6, 1, 7)
13
14 for zdir, x, y, z in zip(zdirs, xs, ys, zs):
15     label = '(%d, %d, %d), dir=%s' % (x, y, z, zdir)
16     ax.text(x, y, z, label, zdir)
17
18 ax.text(1, 1, 1, "red", color='red')
19 ax.text2D(0.05, 0.95, "2D Text", transform=ax.transAxes)
20
21 ax.set_xlim3d(0, 10)
22 ax.set_ylim3d(0, 10)
23 ax.set_zlim3d(0, 10)
24
25 ax.set_xlabel('X axis')
26 ax.set_ylabel('Y axis')
27 ax.set_zlabel('Z axis')
28
29 plt.savefig("mpl-text3d-demo.eps",dpi=200)
30 plt.show()
```

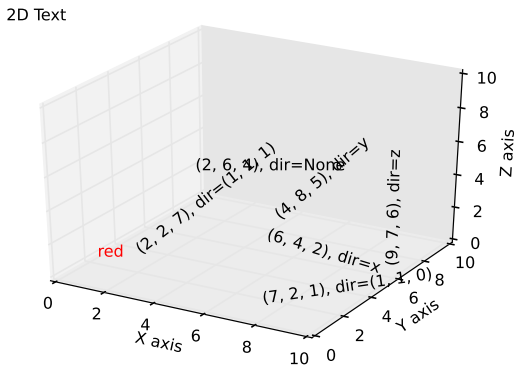


FIGURE 23.20 – mpl-text3d-demo

24

La bibliothèque scipy

La bibliothèque *numpy* ne contient pas toutes les fonctions envisageables. Pour la compléter, il existe la bibliothèque *scipy* qui s'appuie sur *numpy* et qui contient des fonctions de plus haut niveau.

Par exemple, la fonction numérique `exp()` appliquée à une matrice travaille élément par élément. Pour calculer l'exponentielle d'une matrice, il faut utiliser les fonctions `expm()` (approximation de Padé), `expm2()` (diagonalisation) ou `expm3()` (série de taylor) contenues dans la sous-bibliothèque *linalg* de *scipy*.

```
>>> import numpy as np
>>> a=np.arange(16)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a.reshape((4,4))
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a=a.reshape((4,4))
>>> np.exp(a)
array([[  1.00000000e+00,   2.71828183e+00,   7.38905610e+00,
          2.00855369e+01],
       [  5.45981500e+01,   1.48413159e+02,   4.03428793e+02,
          1.09663316e+03],
       [  2.98095799e+03,   8.10308393e+03,   2.20264658e+04,
          5.98741417e+04],
       [  1.62754791e+05,   4.42413392e+05,   1.20260428e+06,
          3.26901737e+06]])
>>> import scipy.linalg as spl    # importation de la sous bibliothèque linalg de
    Scipy
>>> spl.expm(a)
array([[  6.20364637e+12,   7.14131081e+12,   8.07897526e+12,
          9.01663970e+12],
       [  1.79304209e+13,   2.06405557e+13,   2.33506906e+13,
          2.60608254e+13],
       [  2.96571954e+13,   3.41398007e+13,   3.86224059e+13,
          4.31050111e+13],
       [  4.13839700e+13,   4.76390456e+13,   5.38941212e+13,
          6.01491968e+13]])
```

La bibliothèque *scipy* propose également des fonctions de quadrature numérique, d'intégration numérique d'équations différentielles, de résolution numérique de systèmes d'équation, d'optimisation numérique, ...

24.1 Résolution d'équation(s)

Le module *optimize* de *scipy* permet notamment la résolution approchée d'équations ou de systèmes d'équations.
Le programme suivant permet de résoudre l'équation :

$$\sin x = \frac{x}{2} \tag{24.1}$$

par une méthode de Newton avec un point de départ $x_0 = 2$.

newton10.py

```
1  # -*- coding: utf-8 -*-
2
3  import scipy
4  import scipy.optimize
5
6  # f prend en entree une liste de deux elements
7  def f(xy):
8      fval = [0, 0]
9      fval[0] = xy[0]**2 - (xy[1]**2-1)**2
10     fval[1] = (xy[0]-2)**2 -2*xy[1]
11     return fval
12
13 start = [1.8, 2]
14 xysol = scipy.optimize.newton_krylov(f, start, x_tol=1e-7)
15 print (xysol, " : ", f(xysol))
```

Le programme suivant cherche un des huit points d'intersections entre les deux coniques :

$$\begin{cases} x^2 - \left(y^2-1\right)^2 &= 0 \\ \left(x^2-2\right)^2 - 2y^2 &= 0 \end{cases} \tag{24.2}$$

newton11.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy
4  import scipy
5  import scipy.optimize
6
7  def f(x):
8      return numpy.sin(x)-x/2
9
10 start = 2
11 xsol = scipy.optimize.newton_krylov(f, start, f_tol=1e-14)
12 print (xsol)
```

24.2 Quadrature numérique

Le module *scipy.integrate* fournit des fonctions pour la quadrature numérique ainsi que pour l'intégration numérique des équations différentielles.

La fonction `quad(f, a, b)` est à pas adaptatif, et fournit un résultat sous forme de tuple, dont le premier élément est une approximation de $\int_a^b f(x)dx$ où a ou b peuvent être infinis. Le second élément est une estimation de l'erreur absolue commise.

integre.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  import scipy as sp
5  import scipy.integrate as integ
6  import math as m
7
8  def fN(x):
9      return 1/(m.sqrt(2*m.pi))*m.exp(-0.5*x**2)
10
11 print ("De -100 a 1.96 : ", integ.quad(fN,-100,1.96))
12 y, err = integ.quad(fN,-np.inf,1.96)
13 print ("De -oo a 1.96 : %.12f plus ou moins %.2e "%(y, err))
14
15 tabx = np.arange(-100,1.96001,0.01)
16 taby = np.zeros(len(tabx))
17 for i in range(len(tabx)):
18     taby[i] =  fN(tabx[i])
19 print ("De -100 a 1.96 par les trapèzes avec un pas de 1/100 : "),
20 print (integ.trapz(taby, tabx))
```

Le résultat :

```
De -100 a 1.96 :  (0.9750021048517795, 4.2103849945256025e-11)
De -oo a 1.96 : 0.975002104852 plus ou moins 3.62e-09
De -100 a 1.96 par les trapèzes avec un pas de 1/100 :
0.975001150321
```

La fonction `trapz(y,x)` calcule une approximation de $\int_a^b f(x)dx$ par la méthode des trapèzes en se basant sur les vecteurs x et y. Elle sera à pas constant si les points de x suivent une telle répartition, par exemple générée par `numpy.arange()`.

24.3 Intégration numérique des équations différentielles

La fonction `odeint()` permet l'intégration numérique d'une équation différentielle ou d'un système d'équations différentielles du premier ordre. Elle utilise la bibliothèque *odepack*.

L'équation du pendule est :

$$\frac{\partial^2\theta}{\partial t^2} = -\frac{g}{\ell}\sin\theta \tag{24.3}$$

Vous la passez au premier ordre en introduisant une variable $u(t) = \dfrac{\partial\theta}{\partial t}$:

$$\frac{\partial}{\partial t}\begin{pmatrix}\theta\\ u\end{pmatrix} = \begin{pmatrix}u\\ \dfrac{g}{\ell}\sin\theta\end{pmatrix} \tag{24.4}$$

Vous obtenez ainsi la fonction de $\mathbb{R}^2$ dans $\mathbb{R}^2$ qui est la dynamique (ou équation d'état) du système.

odeint1.py

```
1  # -*- coding: utf-8 -*-
```

```
2
3  import numpy as np
4  from scipy.integrate import odeint
5  from pylab import *
6
7  def dyn(x, t, ell, g):
8      (th, u) = x
9      return (u, -g/ell*np.sin(th)-h*u)
10
11 h=0.3
12 x0=(3*np.pi/4, 0)
13 t = np.arange(0, 10, 0.01)
14
15 x = odeint(dyn, x0, t, args=(1, 9.81))
16
17 plot(t, x[:,0], color="blue", linewidth=2.5, linestyle="-")
18
19 ax = gca()
20 ax.spines['right'].set_color('none')
21 ax.spines['top'].set_color('none')
22 ax.xaxis.set_ticks_position('bottom')
23 ax.spines['bottom'].set_position(('data',0))
24 ax.yaxis.set_ticks_position('left')
25 ax.spines['left'].set_position(('data',0))
26
27 xlim(0, 10.3)
28 xticks(range(0, 11))
29
30 ylim(x[:,0].min()*1.1, x[:,0].max()*1.1)
31 yticks(arange(-np.pi, np.pi+1e-12, np.pi/4),
32        [r'$\frac{%i}{4}\pi$'%(i) for i in range(-4,5)])
33
34 # savefig("../img/odeint.png",dpi=200)
35 savefig("odeint.eps",dpi=200)
36
37 show()
```

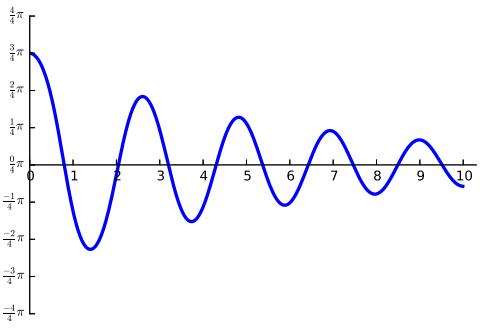


FIGURE 24.1 – Tracé de θ en fonction du temps

La fonction `odeint()` demande au moins trois paramètres : la dynamique de l'équation différentielle, qui doit être une fonction (`dyn` dans l'exemple précédent), le vecteur d'état à l'instant initial ($x0$ dans l'exemple précédent), et les instants pour lesquels on souhaite calculer le vecteur d'état (t dans l'exemple précédent).

Ici, la dynamique demande deux paramètres supplémentaires, que l'on passe via le paramètre `args` de `odeint`. Si l'on dispose du jacobien de la dynamique sous forme de fonction, on peut le fournir à `odeint` par le paramètre `Dfun`, ce qui peut améliorer et/ou accélérer le schéma numérique.

25

Le module sympy

25.1 Introduction

La bibliothèque *sympy* ouvre au langage *Python* la possibilité de réaliser du calcul mathématique symbolique. Il est important de souligner que cette bibliothèque fait actuellement l'objet d'une activité très importante de développement et qu'il est sans doute à attendre dans un avenir proche des améliorations significatives. D'ores et déjà, *sympy* offre un nombre important de fonctionnalités qui le rapproche de logiciels classiques du calcul symbolique comme *Maple* ou *Mathematica*.
Cette section reprend des extraits du tutoriel de *sympy* qui peut être lancé de façon interactive sur le site : docs.sympy.org/0.7.2/tutorial.html.
Voici un premier exemple simple d'application au calcul fractionnel :

```
>>> from sympy import Rational
>>> a = Rational(1,2)
>>> a
1/2
>>> a*2
1
>>> Rational(2)**50/Rational(10)**50
1/888178419700125232338905334472656 25
```

Vous pouvez également écrire :

```
>>> R = Rational
>>> R(1, 2)
1/2
>>> R(1)/2 # R(1) est SymPy Integer et Integer/int donne un Rational
1/2
```

sympy permet aussi l'utilisation d'éléments mathématiques comme π, e ou ∞ :

```
>>> from sympy import pi, E
>>> pi**2
pi**2
>>> pi.evalf()
3.14159265358979
>>> (pi + E).evalf()
5.85987448204884
>>> from sympy import oo
>>> oo > 99999
True
>>> oo + 1
oo
```

25.2 Notion de symbole

Avec *sympy*, si vous souhaitez souhaite travailler avec des variables symboliques comme x, y, z, r, θ, ϕ etc, il est nécessaire de les déclarer explicitement :

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> y = Symbol('y')
```

Certains symboles sont déjà prédéfinis dans la sous-bibliothèque *abc* :

```
>>> from sympy.abc import x, theta
>>> x
x
>>> theta
theta
```

Pour créer des symboles, vous pouvez également utiliser la fonction *var()* :

```
>>> from sympy import symbols, var
>>> a, b, c = symbols('a,b,c')
>>> d, e, f = symbols('d:f')
>>> var('g:h')
(g, h)
>>> var('g:2')
(g0, g1)
```

Voici quelques opérations simples :

```
>>> x+y+x-y
2*x
>>> (x+y)**2
(x + y)**2n
>>> ((x+y)**2).expand()
x**2 + 2*x*y + y**2
```

Lors d'opérations, il est possible de faire prendre à une variable une valeur numérique ou encore la remplacer par une autre expression en utilisant *subs()* :

```
>>> ((x+y)**2).subs(x, 1)
(y + 1)**2
>>> ((x+y)**2).subs(x, y)
4*y**2
>>> ((x+y)**2).subs(x, 1-y)
1
```

Pour améliorer la présentation, on lancera :

```
>>> from sympy import init_printing
>>> init_printing(use_unicode=False, wrap_line=False, no_global=True)
```

25.3 Fractions

Développement de fraction :

```
>>> from sympy import apart
>>> from sympy.abc import x, y, z
>>> 1/( (x+2)*(x+1) )
        1
---------------
(x + 1)*(x + 2)
>>> apart(1/( (x+2)*(x+1) ), x)
    1       1
- ----- + -----
  x + 2   x + 1
>>> (x+1)/(x-1)
x + 1
-----
x - 1
>>> apart(1/( (x+2)*(x+1) ), x)
    1       1
- ----- + -----
  x + 2   x + 1
```

Mise sous dénominateur commun :

```
>>> from sympy import together
>>> together(1/x + 1/y + 1/z)
x*y + x*z + y*z
---------------
     x*y*z
>>> together(apart((x+1)/(x-1), x), x)
x + 1
-----
x - 1
>>> together(apart(1/( (x+2)*(x+1) ), x), x)
        1
---------------
(x + 1)*(x + 2)
```

25.4 Limites

Les limites peuvent être calculées avec la fonction `limit(function, variable, point)` :

```
>>> from sympy import limit, Symbol, sin, oo
>>> x = Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
>>> limit(x, x, oo)
oo
>>> limit(1/x, x, oo)
0
>>> limit(x**x, x, 0)
1
```

25.5 Dérivation

Le calcul des dérivées s'effectue à l'aide de la fonction `diff(func, var)` :

```
>>> from sympy import diff, Symbol, sin, tan
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)
>>> diff(tan(x), x)
   2
tan (x) + 1
```

Dérivation d'ordre supérieur :

```
>>> diff(sin(2*x), x, 1)
2*cos(2*x)
>>> diff(sin(2*x), x, 2)
-4*sin(2*x)
>>> diff(sin(2*x), x, 3)
-8*cos(2*x)
```

25.6 Développements limités

Voici un exemple :

```
>>> from sympy import Symbol, cos
>>> x = Symbol('x')
>>> cos(x).series(x, 0, 10)
     2    4     6      8
    x    x     x      x
1 - -- + -- - --- + ----- + O(x**10)
    2    24   720   40320
>>> (1/cos(x)).series(x, 0, 10)
     2      4       6        8
    x    5*x    61*x    277*x
1 + -- + ---- + ----- + ------ + O(x**10)
    2     24     720     8064
>>> y = Symbol("y")
>>> e = 1/(x + y)
>>> s = e.series(x, 0, 5)
>>> print(s)
1/y - x/y**2 + x**2/y**3 - x**3/y**4 + x**4/y**5 + O(x**5)
```

25.7 Séries

```
>>> from sympy import summation, oo, symbols, log
>>> i, n, m = symbols('i n m', integer=True)
>>> summation(2*i - 1, (i, 1, n))
n**2
>>> summation(1/2**i, (i, 0, oo))
2
>>> summation(1/log(n)**n, (n, 2, oo))
Sum(log(n)**(-n), (n, 2, oo))
>>> summation(i, (i, 0, n), (n, 0, m))
m**3/6 + m**2/2 + m/3
>>> from sympy.abc import x
>>> from sympy import factorial
>>> summation(x**n/factorial(n), (n, 0, oo))
exp(x)
```

25.8 Intégration

Pour le calcul de primitive :

```
>>> from sympy import integrate, erf, exp, sin, log, oo, pi, sinh, symbols
>>> x, y = symbols('x,y')
>>> integrate(6*x**5, x)
x**6
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(exp(-x**2)*erf(x), x)
pi**(1/2)*erf(x)**2/4
```

Pour le calcul d'intégrales définies ou impropres :

```
>>> integrate(x**3, (x, -1, 1))
0
>>> integrate(sin(x), (x, 0, pi/2))
1
>>> integrate(exp(-x), (x, 0, oo))
1
>>> integrate(log(x), (x, 0, 1))
-1
```

25.9 Équations différentielles

```
>>> from sympy import Function, Symbol, dsolve
>>> f = Function('f')
>>> x = Symbol('x')
>>> f(x).diff(x, x) + f(x)
f(x) + Derivative(f(x), x, x)
>>> dsolve(f(x).diff(x, x) + f(x), f(x))
f(x) == C1*cos(x) + C2*sin(x)
```

25.10 Équations algébriques

```
>>> from sympy import solve, symbols
>>> x, y = symbols('x,y')
>>> solve(x**4 - 1, x)
[1, -1, -I, I]
>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{x: -3, y: 1}
```

25.11 Algèbre matricielle

```
>>> from sympy import Matrix, Symbol
>>> Matrix([[1,0], [0,1]])
[1, 0]
[0, 1]
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> A = Matrix([[1,x], [y,1]])
>>> A
[1, x]
[y, 1]
>>> A**2
[x*y + 1,     2*x]
[    2*y, x*y + 1]
```

26

Petit dico scientifique anglais -> français

anglais	français
almost	presque
analysis	analyse
to affect	affecter
to approximate	approcher
arbitrary	arbitrairement
on the assumption that	dans l'hypothèse où
background	arrière-plan
ball	boule
to begin	commencer
beyond	au delà
bottom	bas
bound	borne
bounded	borné(e)
to call	appeler
to choose	choisir
to clear	mettre au clair
colormap	table des couleurs
data	données
decreasing	strictement décroissant(e)
digit	chiffre
to display	afficher
to draw	dessiner
each	chaque
eigenvalue, vector	valeur, vecteur propre
even	pair
ever	toujours, indéfiniment
field	corps, champ
for all	pour tout(e)
function	fonction
generating function	fonction génératrice
generating sequence	suite génératrice
to get	obtenir

anglais	français
increasing	strictement croissant(e)
induced map	application induite
kernel	noyau
key idea	idée principale
label	étiquette
large	grand
least squares	moindres carrés
least squares approximation	approximation en moyenne quadratique
left	gauche
less than	strictement inférieur(e) à
link	lien
map, mapping	application
the most	le (la) plus
to need	avoir besoin
negative	strictement négatif(ve)
non decreasing	croissant(e)
non increasing	décroissant(e)
non negative	positif(ve)
non positive	négatif(ve)
norm	norme
number	nombre
numeric, numerical	numérique(s)
odd	impair
outcome	résultat
polynomial	polynôme, polynomial
positive	strictement positif(ve)
power	puissance
principle	principe
propertie	propriété
to provide	fournir
provided	pourvu que
Q.E.D.	C.Q.F.D.
quickly	rapidement
quotient space (factor space)	espace quotient
radius, radii	rayon, rayons
rational	rationnel(le)
ready	prêt(e)
remainder	reste
reproducing	reproduisant, régénérateur
right	droit
root	racine (voir zero)
rule	règle
to run	exécuter
sample	échantillon
scalar	scalaire
sequence	suite
series	série(s)
step	pas
to set	placer, présenter
small	petit
space	espace

anglais	français
spaced out	échelonné
spanned by	engendré(e) par
square	carré
stable under	stable par
to start	démarrer
to state	déclarer
such that	tel(le) que
symmetric	symétrique
tick	croix
to tick	cocher
top	haut
unbounded	non borné(e)
unless	à moins que
until	jusqu'à ce que
to vanish	s'annuler
vector space	espace vectoriel
window	fenêtre
zero	zéro (d'une fonction)

TABLE 26.2 – dico scientifique anglais -> français

27

Récapitulatif des méthodes

Un petit lien ici en cas de besoin : http://docs.python.org/3/library/stdtypes.html

27.1 Chaînes

Voici quelques méthodes associées aux chaînes :

`chaine.upper()`	met toute la chaîne *chaine* en majuscule
`chaine.lower()`	met toute la chaîne *chaine* en minuscule
`chaine.capitalize()`	met la première lettre de la chaîne *chaine* en majuscule, le reste en minuscule
`chaine.title()`	met la première lettre de chaque mot de la chaîne *chaine* en majuscule, le reste en minuscule
`chaine.swapcase()`	met les minuscules de la chaîne *chaine* en majuscule et les majuscules en minuscule
`chaine.islower()`	indique si la chaîne *chaine* est en minuscule (True/False)
`chaine.isupper()`	indique si la chaîne *chaine* est en majuscule (True/False)
`chaine.isalpha()`	indique si la chaîne *chaine* contient uniquement des caractères alphabétique (True/False)
`chaine.isdigit()`	indique si la chaîne *chaine* contient uniquement des caractères numériques (True/False)
`chaine.isalnum()`	indique si la chaîne *chaine* contient uniquement des caractères alphanumériques (True/False)
`chaine.replace(old , new)`	remplace les bouts de la chaîne *chaine* old par un new
`"el".join(L)`	concaténer une liste avec l'élément 'el' entre chaque item
`chaine.split("el")`	découpe une chaîne *chaine* en liste en fonction du séparateur "el"
`chaine.count("el")`	compte le nombre d'occurrences de la chaîne *chaine* de caractères passée en argument ("el")
`chaine.splitlines([keepend])`	découpe un texte en liste de lignes.
`chaine.find(s[, déb[,fin]])`	cherche la chaîne "sub"
`chaine.encode([enc[, err]])`	encode la chaîne *chaine*

TABLE 27.1 – méthodes pour les chaînes

27.2 Listes

Différentes méthodes associées aux listes sont récapitulées ci-dessous :

`L1 = [el1 ,el2 , el3]`	crée une liste $L1$
`L2 = list(T)`	transforme un tuple T en une liste $L2$
`L = list(range(10))`	liste de 0 à 9 par pas de 1
`L = list(range(2, 20, 3))`	liste de 2 à 20 (exclue) par pas de 3
`in L`	appartient à une liste L
`L = L1+L2`	Concatène deux listes $L1$ et $L2$
`L.sort()`	trie une liste L
`L.reverse()`	inversion !de listeinverse l'ordre des items d'une liste L
`L.count("el")`	compte combien de fois "el" apparaît dans une liste L
`L.append("el")`	ajoute un élément "el" à la fin d'une liste L
`L.index("el")`	trouver l'index d'un élément "el" de la liste L
`L.remove(element)`	enlève l'élément *element* de la liste L
`L.insert(i,x)`	insère l'élément x dans une liste L avant l'élément à la position i (le reste est décalé)
`L.pop(i)`	extrait l'élément x de position i dans une liste L (le reste est décalé)
`len(L)`	donne la longueur d'une liste L
`sum(L)`	somme les éléments d'une liste L
`L[1:6:3]`	sélectionne du $2^{\text{ème}}$ au $7^{\text{ème}}$ (exclu) éléments par pas de 3.
`L[-2]`	sélectionne les deux derniers éléments d'une liste L
`L[2:]`	sélection de l'élément 2 jusqu'à la fin d'une liste L
`L[::-1]`	sélection d'une liste L mais inversée
`L[:]`	copie d'une liste L
`list(L)`	copie d'une liste L

TABLE 27.2 – méthodes pour les listes

Remarques

- L'instruction `del L[i]` enlève l'élément situé à l'index i de la liste L.
- Si aucun indice n'est spécifié, `L.pop()` renvoie le dernier élément de la liste L. L'élément est aussi supprimé de la liste.

Rappel

Rappel : attention aux copies de listes !

27.3 Tuples

Des méthodes associées aux tuples sont indiquées ici :

`T = (el1 ,el2 , el3)`	définit un tuple T
`T2 = tuple(L)`	transforme la liste L en un tuple $T2$
`var1, var2 = var2, var1`	permute des données
`len(T)`	donne la longueur d'un tuple T
`in T`	appartient au tuple T

TABLE 27.3 – méthodes pour les tuples

28

Mots réservés

mot	utilisation	page
`and`	opérateur logique `ET`	45
`as`	associé à `import` pour utiliser les bibliothèques	297, 302
`assert`	permet de rajouter des contrôles pour le débogage d'un programme	non étudié
`break`	interrompt une boucle `while` ou `for`	66
`class`	utilisée en programmation orientée objet (POO)	non étudié
`continue`	saute l'étape suivante dans une boucle `while` ou `for`	66
`def`	définit une fonction	68, 136
`del`	supprime un ou plusieurs éléments d'une liste à partir de leur index	53
`elif`	instruction conditionnelle `SINON SI`	61
`else`	instruction conditionnelle `SINON`	61
`except`	gestion d'erreurs	80
`False`	booléen `FAUX`	48
`finally`	gestion d'erreur	80
`for`	boucle `POUR`	63
`from`	importation de bibliothèques	302
`global`	définit des variables globales dans une fonction	74
`if`	instruction conditionnelle `SI`	61
`import`	importation de bibliothèques	302
`in`	test d'appartenance	53
`is`	test d'égalité	49
`lambda`	définit une fonction par une expression	136
`None`	objet qui ne vaut "rien"	70
`nonlocal`	gestion des variables (locales et globales)	74
`not`	opérateur logique `NON`	62
`or`	opérateur logique `OU`	45
`pass`	instruction vide	62
`raise`	levée d'exception	82
`return`	valeur de retour d'une fonction	68
`True`	booléen `VRAI`	48
`try`	gestion d'erreur	80
`while`	boucle `TANT QUE`	65
`with`	simplification d'écriture	non étudié
`yield`	remplace `return` dans un générateur	non étudié

TABLE 28.1 – Mots réservés

Partie 6

Listes

Table des matières

Figures et tables

Table des figures

Liste des tableaux

Liste des algorithmes

Liste des programmes

Index

Index

Index Python

Symbols

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

X

Commandes Python

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z