# Chapitre 5 : Utilisation de modules et bibliothèques

D'une manière générale, afin de rendre le programme plus court et plus lisible, on évite de réécrire ce qui existe déjà. Pour cela, on favorise l'utilisation de fonctions (sous-parties d'un programme) qui pourront être appelées à plusieurs reprises dans le programme. On favorise ainsi un script plus condensé et une relecture plus aisée.

## 1. Modules

### *1.1.Définition et commandes élémentaires*

Lorsque plusieurs fonctions traitent d'une même thématique, on peut les regrouper dans un fichier ( `*.py` ) appelé module.

L'importation de l'ensemble des fonctions du module (par exemple, le module `math`) s'effectue grâce à la commande : `import math`
Afin de visualiser le contenu du module importé, on utiliser la commande : `dir(math)`
L'appel d'une fonction (par exemple, la fonction cosinus) du module s'effectuera alors de la manière suivante : `math.cos(…)`
On accède à la spécification d'une fonction avec la commande : `help(math.cos)`

On peut souhaiter ne pas importer tout le contenu d'un module afin de ne pas surcharger la mémoire vive de l'ordinateur. Dans ce cas, on écrit : `from math import cos`
Dans ce cas, l'appel de la fonction se fera sans être précédée du nom du module : `cos(…)`

Il est ainsi possible d'importer la totalité des fonctions d'un module de manière à les appeler sans mentionner le nom du module. Pour cela, on écrit par exemple : `from math import *`
Toutefois, il est à préciser que certains modules possèdent des fonctions communes (par exemple, la fonction cosinus appartient au module `math` et au module numpy). Dans le cas où les deux modules sont téléchargés de cette façon, l'appel d'une fonction commune aux deux peut générer un conflit. Cette méthode n'est donc pas recommandée.

Si l'on désire modifier le nom du module qui est à répéter avant chaque appel de fonctions (pour le raccourcir et donc gagner du temps), on peut alors effectuer une importation différente telle que : `import math as m`
Ce qui modifiera la ligne de commande appelant la fonction : `m.cos(…)`

### *1.2.Création de module*

Il est possible de créer son propre module. Pour cela, il suffit de regrouper les fonctions correspondantes dans un fichier ( `*.py` ). Supposons que l'on souhaite créer un module `aires` contenant des fonctions permettant le calcul des aires des formes géométriques usuelles. On peut alors créer le fichier `aires.py` dont le contenu pourrait être le suivant :

```
pi=3.14159

def disque(rayon):
      """rayon de type float est le rayon d'un disque
      renvoie l'aire du disque"""
      return(pi*rayon**2)
```

```
def rectangle(longueur, largeur):
      """largeur et longueur sont du type float
      renvoie l'aire du rectangle"""
      return(largeur*longueur)

def triangle(base,hauteur):
      """base et rayon sont du type float
      renvoie l'aire du triangle"""
      return(base*hauteur/2)
```

Afin que le module soit reconnu et puisse être importé, deux solutions s'offrent à nous :

- Le plus simple est que le fichier créé soit placé dans le dossier de travail dans lequel figureront les fichiers pour lequel l'importation du module sera effectuée via la commande `import aires`
- Placer le fichier créé dans la distribution Python contenant tous les autres modules (solution définitive)

Les chaînes de caractères figurant dans les définitions des fonctions constituent l'aide accessible via la fonction `help`.

### *1.3.Quelques modules intéressants déjà implantés dans Python*

Parmi tous les modules disponibles, citons :

- Le module `math` : contient toutes les fonctions mathématiques et certaines constantes telles que $\pi$
  Exemple : tester l'aide de la commande `hypot`
- Le module `random` : contient certaines fonctions renvoyant des nombres de manière aléatoire
  Exemple : tester l'aide de la commande `randint`
- Le module `turtle` permet de dessiner en programmant.
  Les principales fonctions sont les suivantes :
  - `forward(d)` : avance de `d` pixels
  - `backward(d)` : recule de `d` pixels
  - `left(x)`, `right(x)` : tourne de `x` degrés vers la gauche, vers la droite
  - `reset()` pour effacer
  - `goto(x,y)` : se déplacer au point de coordonnées `x`, `y`
  - `up()` et `down()` pour lever ou abaisser le crayon
  - `color('couleur')` pour sélectionner une couleur
  - `width(e)` pour tracer des traits d'épaisseur `e`
  - `begin_fill()` et `end_fill()` pour remplir une figure fermée avec une couleur

Écrire une fonction `triangle` traçant un triangle équilatéral de côté de longueur `a` passé en argument.

Écrire une fonction `polygone_regulier` traçant un polygone à `n` côtés et de longueur `a` passés en argument. Le tracé devra se faire en bleu, avec une épaisseur de 5 pixels et le polygone rempli en rouge.

## 2. Bibliothèques

### *2.1.Définition*

Une bibliothèque est en général une collection de modules. Il en existe un très grand nombre qui viennent compléter suivant les besoins la bibliothèque standard de Python. Parmi toutes celles-ci, on peut citer les deux plus importantes figurant au programme :

- `matplotlib` pour tout ce qui concerne les graphiques
- `pillow` ou `PIL` pour la manipulation et le traitement d'images

### *2.2.Tracé de graphiques avec Matplotlib*

Parmi tous les modules de cette bibliothèque, nous utiliserons principalement le module `pyplot` importé via la commande :

```
import matplotlib.pyplot as plt
```

Le principe du tracé d'une courbe est le même que si on le faisait sur une feuille de papier. On représente des points appartenant à la courbe à l'aide de leurs coordonnées puis on les relie (ou non) entre eux. Pour cela, on doit donc disposer de deux listes (que nous noterons par la suite `X` et `Y`) contenant pour la première les abscisses des points et pour la seconde leurs ordonnées.

Supposons que nous souhaitions tracer la courbe correspondant à la fonction $f(x)=x^2-5x+1$ sur l'intervalle $[0,1]$ à l'aide de 100 points. Nous devons commencer par créer la fonction permettant le calcul de $f(x)$ puis les listes d'abscisses et d'ordonnées grâce aux commandes :

```
def f(x): # création de la fonction à tracer
      return(x**2-5*x+1)

X=[i*0.01 for i in range(101)] # création de la liste des abscisses
Y=[f(x) for x in X] # création de la liste des ordonnées
```

Un premier tracé peut alors être obtenu de la manière suivante :

```
plt.plot(X,Y) # construction de la courbe
plt.show() # demande l'affichage de la courbe
```

Signalons quelques commandes permettant d'améliorer la présentation du graphique :

```
plt.title("Tracé de la fonction f") # ajout d'un titre au graphique
plt.xlabel("x") # ajout d'un titre à l'axe des abscisses
plt.ylabel("y") # ajout d'un titre à l'axe des ordonnées
plt.grid(True) # affichage un quadrillage
```

La mise en forme de la courbe peut également être modifiée en complétant la ligne de commande correspondant à la construction de la courbe à l'aide des tableaux suivants.

| *Mise en forme* | *Commande* | *racourci* | *Valeurs* |
|---|---|---|---|
| Style de ligne | linestyle | ls | `-` ou `--` ou `-.` ou `:` |
| Epaisseur de ligne | linewidth | lw | Valeur décimale |
| Type de marqueurs | marker | ... | `o` ou `+` ou `,` ou `.` ou `x` ou `1` ou `2` ou `3` ou `4` |
| Taille des marqueurs | markersize | ms | Valeur décimale |
| Couleur des marqueurs | markerfacecolor | mfc | Couleur matplotlib |
| Taille des contours des marqueurs | markeredgewidth | mew | Valeur décimale |
| Couleur des contours des marqueurs | markeredgecolor | mec | Couleur matplotlib |

| *Couleurs* | *Commande* | *racourci* |
|---|---|---|
| Bleu | blue | b |
| Vert | green | g |
| Rouge | red | r |
| Magenta | magenta | c |
| Jaune | yellow | y |
| Noir | black | k |
| Blanc | white | w |

Par exemple :

```
plt.plot(X,Y,color="red",linewidth=2.5,linestyle="--") # construction de la courbe tracée en rouge, avec une épaisseur de 2.5 et avec des traits
```

## 3. Lecture et écriture de fichier texte

### *3.1.Ouverture et fermeture d'un fichier*

L'ouverture et le fermeture d'un fichier texte s'effectue via les fonctions `open` et `close`. Dans le cas de la fonction `open`, le contenu du fichier est placé dans une variable temporaire (jusqu'à la fermeture du fichier) et correspond donc à une attribution.

Lorsqu'on veut ouvrir un fichier, deux éléments sont à préciser dans la fonction `open` :

- Le nom du fichier à ouvrir (ce dernier doit nécessairement se trouver dans le répertoire de travail)
- Le mode d'ouverture retenu : `'w'` pour le mode écriture, `'r'` pour le mode lecture et enfin `'a'` pour le mode ajout lorsqu'on veut ajouter du texte à la fin du fichier.

Par exemple, pour ouvrir un fichier en mode lecture, on aura la commande

```
fic=open('nom du fichier','r')
```

On peut remarquer que la création d'un fichier se fait par une première ouverture en mode écriture (le fichier sera placé suivant le chemin donné au shell).

Il est essentiel de fermer un fichier qui a été ouvert. On écrira alors :
```
fic.close()
```

### 3.2.Écriture

Afin d'écrire dans un fichier, ce dernier doit être ouvert en mode `'w'` ou `'a'`. Dans les deux cas, on utilise alors la fonction `write`.
Exemple : on veut créer le fichier dans lequel on veut écrire « On est bien en MPSI ». On aura alors :
```
fic=open("MPSI.txt","w")
fic.write("On est bien en MPSI")
fic.close()
```

En mode `'a'`, la chaîne de caractères est ajoutée à la suite du texte déjà présent.
Exemple : on veut ajouter au fichier précédent « On est bien à Clemenceau ».
```
fic=open("MPSI.txt","a")
fic.write("\nOn est bien à Clemenceau")
fic.close()
```
Remarque : la syntaxe \n permet un retour à la ligne. Elle peut être placée comme ici au début de la chaîne de caractère ou à la fin.

En revanche, si on ouvre le fichier en mode `'w'`, on remplace le texte déjà présent par une nouvelle chaîne de caractère ; on crée de ce fait une nouvelle version du fichier qui « écrase » la précédente.
```
fic=open("MPSI.txt","w")
fic.write("On a les meilleurs profs (surtout en physique)")
fic.close()
```

### 3.3.Lecture

En mode lecture, plusieurs fonctions sont possibles :
- `read()` : lit tout le fichier depuis le curseur et place l'ensemble dans une variable
- `read(n)` : lit les n caractères du fichier après le curseur et les place dans une variable
- `readline()` : lit la ligne courante et la stocke dans une variable, puis passe à la suivante
- `readlines()` : lit tout le fichier depuis la position du curseur et place l'ensemble dans une variable de type liste pour laquelle chaque élément est une ligne du fichier

On peut effectuer un traitement des chaînes de caractères obtenues à l'aide des fonctions `rstrip` (permet de supprimer le caractère de fin de ligne) et `rsplit(sep)` (couper un chaîne de caractères suivant le délimiter `sep` et renvoie une liste de sous-chaînes).