

# Organiser un programme : Fonctions, modules, fichiers

## 1 Qu'est-ce qu'une fonction

Les fonctions informatiques permettent de structurer un programme informatique compliqué en une architecture d'éléments plus simples.

### 1.1 Analogie avec les fonctions mathématiques

Soit  $f : \mathbb{R} \rightarrow \mathbb{R}$



```
def square(x):  
    y=x*x  
    return y
```

### 1.2 Fonctions de service

Certains téléphones sont équipés de la fonction **réveil**.

Elle déclenche une sonnerie si l'heure de l'horloge correspond à l'heure définie de réveil.



```
def reveil(heure_reveil):  
    if heure_reveil==horloge():  
        sonner()
```

### 1.3 Usage d'une fonction

Le langage python inclut 69 fonctions de bases, en plus des opérateurs de calculs. Ces fonctions utilisent un certains nombres d'arguments et renvoient certains résultats. Pour connaître les arguments et le resultat, il suffit d'utiliser la fonction `help()`. Par exemple :



```
>>> help(abs)  
Help on built-in function abs in module builtins:  
abs(x, /)  
    Return the absolute value of the argument.  
>>> abs(-3.2)  
3.2
```

Quand on veut utiliser une fonction, pour l'appeler, on a besoin des informations suivantes :

- son nom ;
- ses paramètres et leur type ;
- le type de la valeur de retour (s'il y en a une).

Ces informations sont communément appelées la spécification de la fonction. Elles forment la signature de la fonction, petit texte descriptif accessible en utilisant l'aide (`help`).

## 2 Ecrire une fonction

Un programmeur peut aussi créer ses propres fonctions. Il faut prendre le soin de bien les spécifier pour qu'elles soient utilisables par d'autres personnes. L'écriture d'une fonction reprend les principes d'écriture d'un programme auxquels s'ajoutent des éléments complémentaires.

### 2.1 Contexte d'usage : interaction d'une fonction

## 2.2 Elements de rédaction d'une fonction

### 2.2.1 Les éléments du code

Une fonction, sous python, est introduite par le mot clé **def**. Elle possède :

- un **nom** (mêmes règles que pour les noms de variables) ;
- une **spécification**, petit texte placé entre `"""` décrivant la spécification ;
- des **paramètres** ou **arguments**, placés entre parenthèses.
- un **corps** : séquence d'instructions exécutée lors de l'**appel** de la fonction ;
- un retour de résultats (optionnel) introduit par le mot-clé **return**.



```
def nom_de_la_fonction(arguments)
    """Spécification ..."""
    corps_de_la_fonction
    return valeurs
```

Remarques :

- Les fonctions doivent être définies en début de programme, avant d'être utilisées ;
- Le corps de la fonction est indenté d'une tabulation ou 4 espaces ;
- La fin de la fonction est indiqué par l'indentation ou lorsque le mot-clé **return** est atteint.

### 2.2.2 Specification : documentation d'une fonction

Outre le fait qu'il convient de donner un nom explicite à une fonction, il peut vous être demandé de la documenter, c'est-à-dire de spécifier les prérequis sur les arguments formels, leur relation avec le résultat renvoyé, les effets éventuels attendus de la fonction.

Ces informations doivent être placée au début du corps de la fonction, sous forme d'une chaîne de caractère entourée de trois guillemets. Elles sont alors accessible par la commande : **help**(nom de la fonction).



```
def Hypothenuse(x,y):
    """Calcule l'hypothénuse d'un triangle rectangle de petits cotes x et y (nombres)"""
    return (x**2+y**2)**0.5
```

## 3 Usages des variables avec des fonctions

### 3.1 Exemple introductif



```
1  def somme_des_n_entiers_puissance_i(n,i):
2      somme=0
3      for indice in range(n):
4          somme = somme + indice**i
5      return somme
6
7  a=int(input("Combien d'entiers ? "))
8  p=int(input("A quelle puissance ? "))
9  S=somme_des_n_entiers_puissance_i(a,p)
10 print("la somme des n premières entiers à la puissance i est : ",S)
```

## 3.2 Portée des variables

Dès lors qu'on utilise des fonctions, il devient nécessaire de distinguer deux sortes de variables :

- les **variables globales** ont une portée sur l'ensemble du programme : elles sont utilisables (appel et affectation) depuis le programme et les fonctions ;
- les **variables locales** qui ont une portée limitée au corps de la fonction : elles s'effacent de la mémoire à la fin de l'exécution de la fonction.

*Conseil :*

- Utiliser des noms de variables différents pour les variables locales et globales.


## 3.3 Transmission des paramètres d'une fonction

Lors de la définition d'une fonction, les variables qui figurent comme arguments sont appelées **paramètres formels**. Car, ils n'ont pas encore d'existence réelle dans la mémoire. Les paramètres transmis à la fonction lors de son appel sont appelés **paramètres effectifs**. Lors de l'appel d'une fonction :

1. les paramètres formels sont déclarés en tant que variables locales ;
2. ils prennent les valeurs de constantes ou de celles des variables globales précisées par les paramètres effectifs.

## 3.4 Application

On considère le programme suivant :




```
1 def carre(x):
2     y = x**2
3     return y
4 # début du programme
5 a=3
6 print(carre(a))
7 print(a)
```

Décrire l'évolution de l'état du programme.

## 3.5 Fonction de plusieurs arguments, à plusieurs résultats

Une fonction peut nécessiter plusieurs arguments et peut retourner plusieurs valeurs :



```
1 def division_euclidienne(a,b):
2     return a//b,a%b
3 quotient,reste=division_euclidienne(13,3)
4 print(quotient)
5 print(reste)
```

## 4 Les modules

### 4.1 Fonctions prédéfinies

Python s'appuie sur 68 fonctions de bases. Pour résoudre des problèmes complexes, il est possible de s'appuyer sur d'autres fonctions contenues dans des modules qu'il faut importer au cas par cas. Parmi les modules les plus utiles dans ce cours, citons :

- **math** : contient les fonctions usuelles en analyse
- **random** : sert à générer des nombres pseudo-aléatoires
- **numpy** et **scipy** : fournit des outils variés pour le calcul scientifique
- **matplotlib** : permet le tracé de graphes

Il y a différentes possibilités pour charger un module – « Charger » prend ici le sens d'amener dans la mémoire active associée au programme ou à la console en cours d'utilisation. Pour bien faire, il place le chargement en début de programme.

#### 4.1.1 Première méthode : le chargement direct

```
1 from module import *
```

Toutes les fonctions/méthodes du module sont chargées dans la mémoire active. Il est possible de les appeler directement.

On peut ne charger que certaines fonctions du module avec l'instruction :

```
1 from module import fonctions_a_importer
```

Pour plus de commodité et éviter les redondances de nom, il est possible de renommer les fonctions au moment de l'appel avec le mot clé `as` :

```
1 from module import fonction_a_importer as nouveau_nom_fonction
```

#### 4.1.2 Deuxième méthode : le chargement indirect

```
1 import module
```

Toutes les fonctions/méthodes du module sont chargées dans la mémoire active, mais regroupé dans un « paquet » qui porte le nom du module. Une fois le module chargé, pour appeler une fonction en particulier du module, il faut alors utiliser la syntaxe :

```
1 module.fonction_en_particulier
```

Cette méthode permet d'assurer l'absence de redondance des noms utilisés.

#### 4.1.3 Application : interpréter les chargements suivants :

```
from math import *
```

```
import numpy as np
```

```
import scipy as sp
```

```
import scipy.integrate as integr
```

## 5 Données externes : les fichiers

Un fichier est une donnée conservée dans une mémoire de masse (disque dur). Nous ne manipulerons que les fichiers les plus simples : les fichiers texte, reconnaissable grâce au suffixe « `.txt` ».

### 5.1 Présentation

Un fichier externe est un fichier sauvegardé sur un support de masse. Il contient des données préenregistrées selon une convention (*format*). Sous Python, il faut les manipuler en trois temps :

1. Ouvrir le fichier : on précise le nom du fichier avec son chemin d'accès et le mode d'accès (lecture seule ou lecture & écriture) ;
2. Lecture ou écritures des données dans ce fichier ;
3. Fermeture le fichier une fois qu'on a terminé de travailler avec.

### 5.2 Ouverture d'un fichier

Ouvrir un fichier correspond à :

- définir en mémoire une variable correspondant à un lien vers le fichier ;
- signifier au système d'exploitation que le fichier est en cours d'utilisation.

```
mon_fichier=open(Chemin,par)
```

où :

- **Chemin** : nom du fichier précédé de son chemin d'accès : `U:/Python/Projet1/fichier.txt`
- **par** : mode d'ouverture (`r` pour lecture seule, `w` pour lecture et écriture)

### 5.3 Lecture et écriture du contenu textuel

La lecture se fait par méthodes, qui dépendent évidemment du contenu. Dans le cas d'un fichier texte, nous n'en retiendrons que trois :

1. `mon_fichier.read()` : renvoie la chaîne de caractères contenue dans le fichier
2. `mon_fichier.readline()` : renvoie la première ligne contenue dans le fichier. Un nouvel appel de cette méthode renvoie la seconde ligne, etc.
3. `mon_fichier.readlines()` : renvoie une liste de chaînes de caractères où chaque élément est une ligne du contenu du fichier.

L'écriture se fait par la méthode suivante, qui remplace le contenu actuel du fichier par la chaîne `s` :

```
mon_fichier.write(s)
```

### 5.4 Fermeture d'un fichier

Enfin, pour fermer le fichier, la méthode est la suivante :

```
mon_fichier.close()
```