# *Chap 8. Python 3 : Fonctions*

Livre p 57 – Chap 4 Fonctions

## 1. Définition d'une fonction

### a. Introduction

Une fonction est une suite d'instruction que l'on souhaite définir en début de programme afin de l'utiliser plus tard à une ou plusieurs reprises dans le programme.
La fonction porte un nom, « bonjour » par exemple, elle est toujours suivie de parenthèses.
La définition se fait avec la commande « *def* » et se termine par des deux-points « : »
La suite des instructions à utiliser est définie dans un bloc indenté, généralement par une tabulation.

### b. Commentaires

On peut inclure des commentaires encadrés par « """ » qui seront affichés lorsque l'on demandera de l'aide sur la fonction avec la commande « *help*(fct) »

```
def bonjour() :
    """ Un classique """
    print(″Hello world″)
```

### c. Fonctions et procédures

Une fonction renvoie généralement une valeur : un booléen, un nombre, une chaîne de caractères, une liste… C'est l'instruction « *return* » qui le permet.

```
def voyelles() :
    """ Les voyelles """
    return ″aeiouy″
```

Attention : L'instruction « *return* » met alors fin à l'exécution de la fonction
Si l'on n'utilise pas l'instruction *return*, on parle habituellement de procédure plutôt que de fonction.

### d. Paramètres

On peut préciser, dans les parenthèses, un ou plusieurs paramètres séparés par des virgules qui seront utilisés par la fonction.
Lorsque l'on voudra utiliser la fonction, il sera alors nécessaire de préciser la valeur prise par ces paramètres, en les donnant dans le même ordre ou en utilisant leur nom.

```
def moyenne(a, b) :
    m = (a + b)/2
    return m
```

Remarque : Certains paramètres peuvent être optionnels, ils doivent être en dernier dans la liste des paramètres et on leur attribue une valeur par défaut avec un « = ». En l'absence de valeur donnée pour ces paramètres, la valeur par défaut sera utilisée.

### e. Variables locales et globales

Les variables utilisées dans la fonction sont des variables locales, les modifications ne sont plus valables en dehors de la fonction. Si l'on veut modifier les valeurs de variable du programme principal, il faut les indiquer après l'instruction « *global* ».

## 2. Import d'une bibliothèque de fonctions

### a. Introduction

Le langage Python possède de nombreuses instructions de programmation. Certains utilisateurs ont créé des bibliothèques de fonctions spécifiques supplémentaires (ou modules), il faut alors les importer au tout début du programme.

### b. Import direct

| | |
|---|---|
| *from* math *import* sin, cos | Ajoute les fonctions sin() et cos() du module math. |
| *from* random *import* * | Toutes les fonctions du module random sont ajoutées |

Attention : Si une fonction à le même nom qu'une fonction existante, alors la nouvelle la remplace !

### c. Import avec préfixe

*import* math — Toutes les fonctions du module math sont ajoutées, mais il faudra écrire le préfixe « *math.* » avant chaque fonction. Exemple : math.cos(30)

*import* math *as* m — Le préfixe devient « *m.* » . Exemple : m.cos(30)

## 3. Le module random

### a. Introduction

Il n'y a pas de méthode pour générer des nombres réellement aléatoires sur un ordinateur : Un ordinateur ne sait pas lancer des dés !
Un programme informatique est écrit à partir d'un algorithme. Or, le propre d'un algorithme, est de se comporter toujours exactement de la même façon. Difficile donc de créer de l'aléatoire…
Les générateurs de nombres pseudo-aléatoires utilisent généralement des suites récurrentes dont les calculs sont suffisamment compliqués pour qu'on ne puisse pas deviner le terme suivant, tout en respectant certaines lois statistiques.

### b. Réinitialisation

Pour que les valeurs générées ne soient pas toujours les mêmes, la suite de nombres pseudo-aléatoires commence par défaut avec une valeur dépendant du temps machine, en millisecondes. Mais on peut réinitialiser la suite :

*seed*() : Réinitialise le générateur de nombres pseudo-aléatoires à partir du temps machine.
*seed*(n) : Réinitialise le générateur de nombres pseudo-aléatoires à partir de la valeur n.

### c. Nombres (pseudo)aléatoires

*random*() : Crée un nombre à virgule flottante (*float*) aléatoire appartenant à [0.0 ; 1.0]
*randrange*(n) : Crée un nombre entier (*int*) aléatoire appartenant à [0 ; n – 1] (n doit être un entier)
*randint*(a, b) : Crée un nombre entier (*int*) aléatoire appartenant à [a ; b] (a, b doivent être des entiers)

### d. De l'aléatoire sur des chaînes, listes ou tuples

*choice*(objet) : Choisit un élément aléatoire de l'objet cité.
Exemples : *choice*(″aeiouy″) (choisit une voyelle au hasard)
*choice*([″lundi″, ″mardi″, ″mercredi″, ″jeudi″, ″vendredi″, ″samedi″, ″dimanche″])

*sample*(objet, n) : Crée une liste de taille n d'éléments aléatoires de l'objet cité.
Exemple : *sample*(″aeiouy″, 2) (créé une liste de deux voyelles aléatoires différentes)

Remarque : Si un élément est présent plusieurs fois, il peut alors être choisi autant de fois qu'il s'y trouve.

*shuffle*(liste) : Mélange les éléments de la liste nommée.
Attention : La liste initiale est modifiée, cela ne crée pas de nouvelle liste !
Il existe d'autres fonctions présentes dans ce module, elles ont pour la plupart une action sur le mode de distribution des nombres aléatoires générés.

## 4. Le module math

### a. Quelques fonctions utiles

Les fonctions racine carré, exponentielle et logarithme népérien : *sqrt*(), *exp*(), *log*()
Remarque : *log10*() est le logarithme décimal.

### b. Trigonométrie

On retrouve les fonctions classiques de trigonométrie : cos(), sin(), tan(),
ainsi que leurs réciproques : acos(), asin(), atan().
Remarque : pour obtenir la valeur de $\pi$, écrire simplement *pi*.
Attention : L'unité est le radian, mais il existe deux fonctions de conversion degrees() et radians().