Cours de Programmation en Python

1 Introduction

demos
Learning is changing®

Fonctions et modules ?

1 Meilleure organisation du programme
2. Eviter la repetition de code et meilleure reutlisation du code
3. Possibilité de partager les fonctions (via des modules)
4. Le programme principal doit être le plus simple possible

1. Module = fichier « .py »
2. On peut regrouper dans un module les fonctions traitant
des problèmes de même nature ou manipulant le même
type d'objet
3. Pour charger les fonctions d'un module dans un autre
module / programme principal, on utilise la commande
import nom_du_module
4. Les fonctions importées sont chargées en mémoire. Si
collision de noms, les plus récentes écrasent les anciennes.

Fonctions

- Fonction est un bloc d'instructions
- Prend (éventuellement) des paramètres en entrée (non typés)
- Renvoie une valeur ou plusieurs valeurs en sortie

```
def petit (a, b):
    if (a < b):
        d = a
    else:
        d = 0
    return d
```

- def pour dire que l'on définit une fonction
- Le nom de la fonction est « petit »
- Les paramètres ne sont pas typés
- Noter le rôle du ":"
- Attention à l'indentation
- return renvoie la valeur
- return provoque immédiatement la sortie de la fonction

Passage de paramétres

Passage des paramètres par position:

```
print(petit(10, 12))
```

Passage des paramètres par nom :

```
print(petit(a=10,b=12))
print(petit(b=12,b=10))
```

Paramètres par defaut

- Affecter des valeurs aux paramètres dès la définition de la fonction
- Si l'utilisateur omet le paramètre lors de l'appel, cette valeur est utilisée
- Si l'utilisateur spécifie une valeur, c'est bien cette dernière qui est utilisée
- Les paramètres avec valeur par défaut doivent être regroupées en dernière position dans la liste des paramètres

```
def ecart(a,b,epsilon = 0.1):
    d = math.fabs(a - b)
    if (d < epsilon):
        d = 0
    return d

ecart(a=12.2, b=11.9, epsilon = 1) #renvoie 0
ecart(a=12.2, b=11.9) #renvoie 0.3
```

Variables et scopes

1. Les variables définies localement dans les fonctions sont uniquement visibles dans ces fonctions.
2. Les variables définies (dans la mémoire globale) en dehors de la fonction ne sont pas accessibles dans la fonction
3. Elles ne le sont uniquement que si on utilise un mot clé spécifique

```
#fonction
def modif_1(v):
    x = v

#appel
x = 10
modif_1(99)
print(x) ➔ 10
```

x est une variable locale, pas de répercussion

```
#fonction
def modif_2(v):
    x = x + v

#appel
x = 10
modif_2(99)
print(x)
```

x n'est pas assignée ici, l'instruction provoque une ERREUR

```
#fonction
def modif_3(v):
    global x
    x = x + v

#appel
x = 10
modif_3(99)
print(x) ➔ 109
```

On va utiliser la variable globale **x**. L'instruction suivante équivaut à x = 10 + 99

Modules

Modules

• Un module est un fichier « .py » contenant un ensemble de variables, fonctions et classes que l'on peut importer et utiliser dans le programme principal (ou dans d'autres modules).

• Le mot clé import permet d'importer un module

• Des modules standards prêts à l'emploi sont livrés avec la distribution Python. Ex. random, math, os, hashlib, etc.

Modules

```
import math, random


random.seed(None)
value = random.random()

#calculer le carré de
#son logarithme
logv = math.log(value)
abslog = math.pow(logv,2.0)

#affichage
print(abslog)
```

Préfixer la fonction à utiliser par le nom du module

Pour un module personalisé Il suffit de créer un fichier nom_module.py, et d'y implémenter les fonctions à partager. De l'importer ensuite à l'aide du mot clef import

Rq : mettre le module dans le même dossier de votre projet python