Programmation en Python
(V)

Akka Zemmari
Hervé Hocquard
herve.hocquard@u-bordeaux.fr

LaBRI, Université de Bordeaux - CNRS

17 septembre, 2023

université de BORDEAUX

Plan

Ce que nous verrons dans ce cours :

- Opérateurs de base
- Types de variables
- Nombres
- Chaines de caractères
- Listes
- Tuples
- Dictionnaire
- Branchements conditionnels
- Boucles
- Fonctions
- Modules
- Fichiers I/O
- Exceptions
- Classes et Objets

université de BORDEAUX

Procédures et fonctions

Les procédures et les fonctions, pour quoi faire ?

- Meilleure organisation du programme
  $\rightarrow$ lisibilité et maintenance
- Éviter la redondance
  $\rightarrow$ factorisation de code
- Possibilité de partager les fonctions (via des modules)
- $\Rightarrow$ Le programme principal doit être le plus simple possible

université de BORDEAUX

Définition des fonctions

- Une fonction :
  - est un bloc d'instructions
  - Prend (éventuellement) des paramètres en entrée
  - Renvoie (ou pas) une valeur en sortie (ou plusieurs valeurs.
- Une fonction sans return est une procédure.
- Exemple :

```
1    def petit(a, b):
2        if a<b:
3            c = a
4        else
5            c = 0
6        return c
```

- Syntaxe :

```
1    def nom_de_la_fonction(param[, param]):
2        instructions
3        [return valeur]
```

Appel des fonctions

- Passage de paramètres par position:

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

  → La fonction renvoie 10

- Passage par nom

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

  → La fonction renvoie 10

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

  → La fonction renvoie 10

- Attention ...

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

  → La fonction renvoie 0

université de BORDEAUX

Appel des fonctions

Exemple :

```
1     #On définit la fonction :
2     def petit(a, b):
3         if a<b:
4             c = a
5         else
6             c = 0
7         return c
8     #On saisit les valeurs de x et de y :
9     x = int(input("x : "))
10    y = int(input("y : "))
11    #On appelle la fonction :
12    res = petit(a=x,b=y)
13    #affichage du résultat :
14    print("résultat : " + str(res))
15    print("pause ...")
```

Valeur par défauts

Paramètres par défaut

- 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 en dernière position dans la liste des paramètres

université de BORDEAUX

Valeur par défauts

Exemple :

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

A l'exécution :

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

Renvoyer plusieurs valeurs avec return

```
1  def extreme(a,b):
2      if (a < b):
3          return a,b
4      else:
5          return b,a
```

A l'appel :

```
1  x = 10
2  y = 15
3  vmin,vmax = extreme(x,y)
4  print(vmin,vmax)
```

On obtient :

```
vmin =10 et vmax=15
```

Question : Que se passe-t-il si nous ne mettons qu'une variable dans la partie gauche de l'affectation ?

```
1  v = extreme(x,y)
2  print(v)
3  #quel type pour v ?
4  print(type(v))
```

On obtient :

```
<class 'tuple'>
```

En fait, v vaut (10,15)

Visibilité des variables

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

```
1  #fonction
2  def modif_1(v):
3      x = v
4  #appel
5  x = 10
6  modif_1(99)
7  print(x) #--> 10
```

x est une variable locale, pas de répercussion

```
1  #fonction
2  def modif_2(v):
3      x = x + v
4  #appel
5  x = 10
6  modif_2(99)
7  print(x)
```

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

```
1  #fonction
2  def modif_3(v):
3      global x
4      x = x + v
5  #appel
6  x = 10
7  modif_3(99)
8  print(x) #--> 109
```

On va utiliser la variable globale x.

université de BORDEAUX

Les 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
- C'est un pas supplémentaire vers la modularité : un module maximise la réutilisation et facilite le partage du code

Les 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
- C'est un pas supplémentaire vers la modularité : un module maximise la réutilisation et facilite le partage du code

Modules standards

- Des modules standards prêts à l'emploi sont livrés avec la distribution Python.
  Ex. `random`, `math`, `os`, `hashlib`, etc.
- Ils sont visibles dans le répertoire « `Lib` » de Python
- Liste complète : `https://docs.python.org/3/library/`

université de BORDEAUX

Exemple d'utilisation de modules standards

```
1  #importer les modules #math et random
2  import math, random
3  #génerer un nom réel compris entre 0 et 1
4  random.seed(None)
5  value = random.random()
6  #calculer le carré de son logarithme
7  logv = math.log(value)
8  abslog = math.pow(logv,2.0)
9  #affichage
10 print(abslog)
```

université de BORDEAUX

Exemple d'utilisation de modules standards

```
1  #définition d'alias
2  import math as m, random as r
3  #utilisation de l'alias
4  r.seed(None)
5  value = r.random()
6  logv = m.log(value)
7  abslog = m.pow(logv,2.0)
```

L'alias permet d'utiliser des noms plus courts dans le programme.

université de BORDEAUX

Exemple d'utilisation de modules standards

```
1  #définition d'alias
2  import math as m, random as r
3  #utilisation de l'alias
4  r.seed(None)
5  value = r.random()
6  logv = m.log(value)
7  abslog = m.pow(logv,2.0)
```

L'alias permet d'utiliser des noms plus courts dans le programme.

```
1  #importer le contenu des modules
2  from math import log, pow
3  from random import seed, random
4  #utilisation directe
5  seed(None)
6  value = random()
7  logv = log(value)
8  abslog = pow(logv,2.0)
```

Permet de désigner nommément les fonctions à importer.
Plus besoin de préfixer les fonctions avant de les appeler ...

université de BORDEAUX

Création d'un module personnalisé

Il suffit de créer un fichier nom_module.py, et d'y implémenter les fonctions à partager :

```
1  #taxe à 10%
2  def pttc_reduit(p):
3      return p * 1.1
4
5  #taxe à 20%
6  def pttc_normal(p):
7      return p * 1.2
8
9  #taxe à 5.5%
10 def pttc_alimentaire(p):
11     return p * 1.055
```

université de BORDEAUX

Importation d'un module personnalisé

```
1  #Importation du module
2  import tva
3  #Programme principal
4  #Saisie et lecture du prix HT
5  pht = int(input("prix : "))
6  #Affichage du prix TTC
7  pttc = tva.pttc_normal(pht)
8  print(pttc)
```

université de BORDEAUX

Importation d'un module personnalisé

```
1  #Importation du module
2  import tva
3  #Programme principal
4  #Saisie et lecture du prix HT
5  pht = int(input("prix : "))
6  #Affichage du prix TTC
7  pttc = tva.pttc_normal(pht)
8  print(pttc)
```

Python cherche automatiquement le module dans le `search path` :

- le dossier courant
- les dossiers listés dans la variable d'environnement `PYTHONPATH`
- les dossiers automatiquement spécifiés à l'installation.

université de BORDEAUX

Module personnalisé : documentation

Documentez votre code, pensez aux autres.

```
"""Module pour le calcul des prix TTC
"""
#taxe à 10%
def pttc_reduit(p):
    """tva intermédiaire
    """
    return p * 1.1
#taxe à 20%
def pttc_normal(p):
    """tva normale
    """
    return p * 1.2
#taxe à 5.5%
def pttc_alimentaire(p):
    """tva produits alimentaires
    """
    return p * 1.055
```

Module personnalisé : documentation

```
>>import tva
>>help(tva)
Help on module TVA:

NAME
    tva

DESCRIPTION
    Module pour le calcul des prix TTC

FUNCTIONS
    pttc_reduit(p):
        tva intermédiaire

    pttc_normal(p):
        tva normale

    pttc_alimentaire(p):
        tva produits alimentaires
FILE
    ....tva.py
```