Programmation en Python
(IV)

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

Une liste est une "sorte de " collection

- Une collection permet de mettre plusieurs valeurs dans une seule variable.
- Une collection permet de regrouper toutes les valeurs dans un seul et même paquet.

```
1       country = ['USA','Italy','Maroc']
2       greeting = ['Hello','Ciao','Salam']
```

université de BORDEAUX

Liste de constantes

- Une liste de constantes est entourée de "[]" et les éléments sont séparés par des ",".
- Un élément de liste peut être tout objet Python. En particulier, il peut être une liste.
- Une liste peut être vide.

```
1  print([1, 24, 76])
2  print(['red', 'yellow', 'blue'])
3  print(['red', 24, 98.6])
4  print([ 1, [5, 6], 7])
5  print([])
```

À l'exécution, on obtient :

```
[1, 24, 76]
['red', 'yellow', 'blue']
['red', 24, 98.6]
[1, [5, 6], 7]
[]
```

université de BORDEAUX

Accès aux éléments de la liste

Comme pour les chaînes de caractères, on peut accéder à un élément de la liste en donnant son indice entre "[]".

Jean	Pierre	Maris
0	1	2

```
1  amis = ['Jean', 'Pierre', 'Maris']
2  print(amis[1])
```

À l'exécution, on obtient :

```
Pierre
```

université de BORDEAUX

Les listes sont modifiables

- Les chaînes sont "immuables"
  nous ne pouvons pas modifier le contenu d'une chaîne - nous devons créer une nouvelle chaîne pour apporter des modifications.

université de BORDEAUX

Les listes sont modifiables

- Les chaînes sont "immuables"
  nous ne pouvons pas modifier le contenu d'une chaîne - nous devons créer une nouvelle chaîne pour apporter des modifications.

- Les listes sont "modifiables"
  nous pouvons changer un élément d'une liste en utilisant l'opérateur d'index.

université de BORDEAUX

Les listes sont modifiables

```
1  fruit = 'Pomme'
2  fruit[0] = 'b'
```

À l'exécution, on obtient :

```
Traceback
TypeError: 'str' object does not support item assignment
```

```
1  x = fruit.lower()
2  print(x)
```

À l'exécution, on obtient :

```
pomme
```

université de BORDEAUX

Les listes sont modifiables

```
1  lotto = [2, 14, 26, 41, 63]
2  print(lotto)
```

À l'exécution, on obtient :

```
[2, 14, 26, 41, 63]
```

```
1  lotto[2] = 28
2  print(lotto)
```

À l'exécution, on obtient :

```
[2, 14, 28, 41, 63]
```

université de BORDEAUX

Les listes

Longueur d'une liste :

- La fonction `len()` prend une liste en argument et retourne le nombre d'éléments qu'elle contient.
- En fait, la fonction s'applique à toute collection...

```
1  greet = 'Hello Bob'
2  print(len(greet))
```

À l'exécution, on obtient :

```
9
```

```
1  x = [ 1,2,'joe',99]
2  print(len(x))
```

À l'exécution, on obtient :

```
4
```

université de BORDEAUX

Les listes

La fonction `range` :

- La fonction `range()` retourne une liste de nombres allant de zéro à la valeur du paramètre -1.

```
1  print(range(4))
```

À l'exécution, on obtient :

```
[0,1,2,3]
```

```
1  friends = ['Joseph', 'Glenn', 'Sally']
2  print(len(friends))
```

A l'exécution, on obtient :

```
3
```

université de BORDEAUX

Tester l'appartenance

- Deux opérateurs pour savoir si un élément fait partie d'une liste : `in` et `not in`.
- Il s'agit d'opérateurs logiques, donc qui retournent `True` ou `False`.

```
1  some = [1, 9, 21, 10, 16]
2  9 in some
```

À l'exécution, on obtient :

```
True
```

```
1  5 not in some
```

À l'exécution, on obtient :

```
True
```

université de BORDEAUX

Liste triée

- Une liste contient des éléments. Les éléments sont stockés dans l'ordre où ils ont été insérés.
- Une liste peut être triée grâce à la méthode `sort`.

```
1  friends = [ 'Joseph', 'Glenn', 'Sally' ]
2  friends.sort()
3  print(friends)
```

À l'exécution, on obtient :

```
['Glenn', 'Joseph', 'Sally']
```

```
1  print(friends[1])
```

À l'exécution, on obtient :

```
Joseph
```

université de BORDEAUX

Fonctions prédéfinies pour les listes

```
1 nums = [3, 41, 12, 9, 74, 15]
2 print(len(nums))
```

```
6
```

```
1 print(max(nums))
```

```
74
```

```
1 print(min(nums))
```

```
3
```

```
1 print(sum(nums))
```

```
154
```

```
1 print(sum(nums)/len(nums))
```

```
74
```

université de BORDEAUX

Les tuples

- Les tuples sont une collection assez semblable aux listes.
- Les indices commencent à 0.
- Mais attention : les tuples sont immuables...

```
x = ('Glenn', 'Sally', 'Joseph')
print(x[2])
```

```
Joseph
```

```
y = ( 1, 9, 2 )
print(y)
```

```
(1, 9, 2)
```

```
print(max(y))
```

```
9
```

université de BORDEAUX

Les tuples

A ne pas faire :

```
x = (3, 2, 1)
x.sort()
```

```
Traceback:AttributeError:  'tuple' object has no attribute 'sort'
```

```
x.append(5)
```

```
Traceback:AttributeError:  'tuple' object has no attribute 'append'
```

```
x.reverse()
```

```
Traceback:AttributeError:  'tuple' object has no attribute 'reverse'
```

université de BORDEAUX

Les dictionnaires

- Les types composites (chaînes de caractères, listes et tuples) considérés jusqu'à maintenant étaient tous des séquences, i.e. des suites ordonnées d'éléments.

Les dictionnaires

- Les types composites (chaînes de caractères, listes et tuples) considérés jusqu'à maintenant étaient tous des séquences, i.e. des suites ordonnées d'éléments.
- On peut accéder à un élément d'une séquence à partir de sa position.

Les dictionnaires

- Les types composites (chaînes de caractères, listes et tuples) considérés jusqu'à maintenant étaient tous des séquences, i.e. des suites ordonnées d'éléments.
- On peut accéder à un élément d'une séquence à partir de sa position.
- Un dictionnaire ressemble à une liste et est modifiable mais n'est pas une séquence car les éléments enregistrés ne sont pas disposés dans un ordre immuable.

université de BORDEAUX

Les dictionnaires

- Les types composites (chaînes de caractères, listes et tuples) considérés jusqu'à maintenant étaient tous des séquences, i.e. des suites ordonnées d'éléments.
- On peut accéder à un élément d'une séquence à partir de sa position.
- Un dictionnaire ressemble à une liste et est modifiable mais n'est pas une séquence car les éléments enregistrés ne sont pas disposés dans un ordre immuable.
- On peut accéder à un élément d'un dictionnaire à partir d'une clé. Cette clé peut être une chaîne, un nombre ou même d'un type composite sous certaines conditions. On ne peut pas modifier les clés d'un dictionnaire.

Les dictionnaires

```
1  traduction = {}          #Dictionnaire vide
2  print(traduction)
```

```
{}
```

```
1  traduction["mouse"] = "souris"      #insertion d'éléments à l'aide de
2  traduction["keyboard"] = "clavier" #paires clé-valeur
3  print(traduction)
```

```
{'mouse': 'souris', 'keyboard': 'clavier'}
```

université de BORDEAUX

Les dictionnaires

On peut aussi créer un dictionnaire comme suit :

```
1  D = {"Duclos" : "Pierre", "Perron" : "Luc"}
2  F, G = {}, {5: "Mauve", 2: "Rouge"}
3  print(D, F, G)
```

```
{'Perron': 'Luc', 'Duclos': 'Pierre'} {} {2: 'Rouge', 5: 'Mauve'}
```

À noter que nous pouvons utiliser en guise de clés n'importe quel type de données non modifiables : des entiers, des réels, des chaînes de caractères et même des tuples.

Les dictionnaires

Contrairement à une liste, nous ne faisons pas appel à une méthode particulière telle que `append()` pour ajouter un nouvel élément à un dictionnaire. Il suffit de créer une nouvelle paire clé-valeur.

```
1 print(traduction)
```

```
{'mouse': 'souris', 'keyboard': 'clavier'}
```

```
1 traduction["computer"] = "ordinateur"
2 print(traduction)
```

```
{'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}
```

université de BORDEAUX

Supprimer les éléments d'un dictionnaire

```
del traduction["mouse"]
print(len(traduction))
```

```
2
```

```
print(traduction)
```

```
{'computer': 'ordinateur', 'keyboard': 'clavier'}
```

```
traduction.pop('computer')
#Supprime et retourne l'entrée.
```

```
'ordinateur'
```

```
traduction.clear()  #Supprime toutes les entrées.
print(traduction)
```

```
{}
```

```
del traduction      #Supprime un dictionnaire.
print(traduction)
```

```
Traceback (most recent call last):
    File "<pyshell#44>", line 1, in <module>
        print traduction
    NameError: name 'traduction' is not defined
```

université de BORDEAUX

Accès aux valeurs d'un dictionnaire

`keys()` : renvoie la liste des clés utilisées dans le dictionnaire.
Cela permet de parcourir un dictionnaire et d'accéder à ses valeurs.

```
1  dictionnaire = {'rouge': 'red', 'vert': 'green', 'bleu': 'blue'}
2  for cle in dictionnaire.keys():
3      print ('clé = %s, valeur = %s' % (cle, dictionnaire[cle]))
```

```
clé = bleu, valeur = blue
clé = vert, valeur = green
clé = rouge, valeur = red
```

université de BORDEAUX

Accès aux valeurs d'un dictionnaire

On peut aussi accéder isolément aux éléments d'un dictionnaire en spécifiant la clé souhaitée entre crochets :

```
1 dictionnaire = {'rouge': 'red', 'vert': 'green', 'bleu': 'blue'}
2 print('clé = %s, valeur = %s' % ('vert', dictionnaire['vert']))
```

```
clé = vert, valeur = green
```

université de BORDEAUX

Accès aux valeurs d'un dictionnaire

Si nous essayons d'accéder à une donnée à l'aide d'une clé qui ne figure pas dans le dictionnaire, nous obtenons une erreur.

```
1  dictionnaire['noir']
```

```
Traceback (most recent call last):
    File "<pyshell#14>", line 1, in <module>
        dictionnaire['noir']
KeyError: 'noir'
```

université de BORDEAUX

Fonctions et méthodes permettant de manipuler des dictionnaires

- `D.values()` : renvoie la liste des valeurs utilisées dans le dictionnaire `D`.
- `D.has_key()` : renvoie `True` (`False`) si le dictionnaire `D` (ne) contient (pas) la clé passée en paramètre.
- `D.items()` : renvoie une liste équivalente de tuples à partir d'un dictionnaire `D`.
- `D.copy()` : effectue une vraie copie d'un dictionnaire `D` au lieu d'un alias.

université de BORDEAUX

Dictionnaire : Un exemple

Construction d'un histogramme à l'aide d'un dictionnaire :

```
1  texte = "Les dictionnaires constituent un outil très élégant pour construire des histogrammes."
2  caracteres = {}
3  for c in texte:
4      caracteres[c] = caracteres.get(c, 0) + 1
5  print(caracteres)
```

```
{'a': 3,' ': 10,'c': 3,'e': 6, 'd': 2, 'g': 2, 'p': 1, 'i': 7, 'è': 1, 'm': 2,
'L': 1, 'o': 6, 'n': 7, 'é': 2, 's': 8, 'r': 6,'u': 5, 't': 9, 'h': 1,
'.': 1, 'l': 2}
```

université de BORDEAUX

Dictionnaire : Un exemple

```
1  caracteres_tries = caracteres.items()
2  caracteres_tries.sort()
3  print(caracteres_tries)
```

```
[(' ', 10), ('.', 1), ('L', 1), ('a', 3), ('c', 3), ('d', 2), ('e', 6),
('g', 2), ('h', 1), ('i', 7), ('l', 2), ('m', 2), ('n', 7), ('o', 6),
('p', 1), ('r', 6), ('s', 8), ('t', 9), ('u', 5), ('è', 1), ('é', 2)]
```

université de BORDEAUX

Dictionnaire : Autres fonctions

- `type()` : renvoie le type de la variable.
- `str()` : renvoie une chaîne de caractères représentant le dictionnaire.
- `dict()` : renvoie un dictionnaire issu par exemple d'une séquence où ses éléments doivent être associés deux à deux. Le premier élément constituera une nouvelle clé et le second sa valeur associée.

```
1 dict([[1,2], [6,7], [3,9]])
2 dict([(1,2), (6,7), (3,9)])
3 dict(((1,2), (6,7), (3,9)))
```

À l'exécution :

```
{1: 2, 3: 9, 6: 7}
{1: 2, 3: 9, 6: 7}
{1: 2, 3: 9, 6: 7}
```

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.

```
1  """Module pour le calcul des prix TTC
2  """
3  #taxe à 10%
4  def pttc_reduit(p):
5      """tva intermédiaire
6      """
7      return p * 1.1
8  #taxe à 20%
9  def pttc_normal(p):
10     """tva normale
11     """
12     return p * 1.2
13 #taxe à 5.5%
14 def pttc_alimentaire(p):
15     """tva produits alimentaires
16     """
17     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
```