UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Licence informatique & vidéoludisme
Semestre 1

Méthodologie de la programmation

Chapitre 1

Programmation impérative avec python

1. Retours notions de programmation

2. Les fonctions en Python

Tuples

- Python permet de manipuler des paires, des triplets, des quadruplets, ...
- La syntaxe est de séparer les expressions par des virgules.
- Exemples :
  - nom, age = "Sam", 24
  - a, b = b, a

! Attention 1,23 est la paire composée de 1 et 23, pas le nombre 1.23.

Combiner des conditions

- pour renvoyer `True` si `expr1` et `expr2` sont vraies *en même temps*, on utilise l'opérateur "`and`"
- pour renvoyer `True` si `expr1` ou `expr2` sont vraies on utilise l'opérateur "`or`" (caractère « *pipe* »).

```
1 x = 12
2 if x > 6 and x < 15 :
3         print x
4 else :
5         print("le nombre n'est pas compris entre 6 et 15)
```

```
1 x = 12
2 if x > 6 or x < 15 :
3         print x
4 else :
5         print("le nombre ....?")
```

Création facile de boucle

- on peut créer une boucle `for` facilement grâce à la fonction `range`

```
1 for \item in range(100):
2         print(i) # affiche tous les nombres entre 0 et ?? (testez)
```

Parcours de listes et de dictionnaires

```
1 l = [1,2,3] # liste
2 for element in l:
3         print(element)
4
5
6 d = {'age':'18', 'prenom':'Anna'} # dictionnaire
7 for k,v in d.items():
8         print(k, v)
```

1. Retours notions de programmation

2. Les fonctions en Python

Rappel

- Une fonction est un « sous-programme ».
- Elles permettent de réutiliser plusieurs fois le même code à différents endroits.
- Une fonction reçoit un ou des arguments (ou paramètres) et **renvoie un résultat**.

```
fonction calculer\_age (annee\_naissance):
        renvoyer annee\_courante - annee\_naissance
```

```
fonction fact (n):
        resultat ← 1
        tant que n > 1 faire:
                resultat ← n * resultat
                n ← n - 1
        renvoyer resultat
```

print *vs.* return

Comparer ces deux fonctions :

```
1 def mult2\_v1(n):
2         return n*2
```

```
1 def mult2\_v2(n):
2         print(n*2)
```

print *vs.* return

Comparer ces deux fonctions :

```
1 def mult2\_v1(n):
2         return n*2
```

```
1 def mult2\_v2(n):
2         print(n*2)
```

Quel affichage produit le code suivant ?

```
1 def add(a, b):
2     result = a + b
3     return result
4 add(2, 2)
```

Comment faire afficher « 4 » à l'écran ?

print *vs.* return

Bonne pratique pour afficher le résultat d'une fonction

```
# Cette solution n'est PAS recommandée
def add(a, b):
    result = a + b
    print(result)
add(2, 2)
```

```
# Cette solution est recommandée :
def add(a, b):
    result = a + b
    return result
print(add(2, 2))
# vous pouvez aussi stocker le résultat
```

Portée d'une variable

= **partie du code dans laquelle on peut y accéder.**

- Les variables qui sont initialisées en dehors d'une fonction sont globales (accessibles partout).
- Les variables qui sont initialisées dans une fonction sont locales, elles n'existent que dans cette fonction.
- L'affectation n'est pas une modification « en place » , elle crée une variable locale.

```
1  a = 5 # global
2
3  def fct(arg):
4          b = "foo"
5          r = b * (a + arg) # ic\item a est accessible
6          a = 10
7          return r
8
9  # b et r ne sont plus accessibles ici
10 # ici a vaut 5
```

Passage des arguments

En Python les arguments sont passés aux fonctions par référence.

- si on fait une modification en place de ces variables, alors **elles seront affectées en dehors de la fonction** aussi.
- L'**affectation** n'est pas une modification « en place », elle crée une variable locale.

```
1  def f(lst1, lst2):
2          lst1 = [1,2,3]
3          lst2.append(13)
4          return
5
6  a = [0,1,0,1]
7  b = [10,11,12]
8  print(a)
9  print(b)
10 f(a, b)
11 print(a)
12 print(b)
```

Passage des arguments

En Python les arguments sont passés aux fonctions par référence.

- si on fait une modification en place de ces variables, alors **elles seront affectées en dehors de la fonction** aussi.
- L'affectation n'est pas une modification « en place », elle crée une variable locale.

```
1  def f(lst1, lst2):
2          lst1 = [1,2,3] # ceci est une affectation
3          lst2.append(13)
4
5  a = [0,1,0,1]
6  b = [10,11,12]
7  print(a) # a = [0,1,0,1]
8  print(b) # b = [10,11,12]
9  f(a, b)
10 print(a) # a = [0,1,0,1]
11 print(b) # b = [10,11,13]
```

Appel de fonction

- Les appel de fonction sont de la forme `nom_de_la_fonction(argument_1, argument_2, ...)`
- Les arguments peuvent être tous types d'expressions.
- Quand une fonction attend plusieurs arguments, ceux-ci sont séparés par des virgules.

« *Built in* functions »

Certaines fonctions existent déjà

« *Built in* functions »

Certaines fonctions existent déjà

- Certaines fonctions **mathématiques** :
  - `round`(2.35)
  - `max`(a+3, 24)
- Les fonctions de **cast** :
  - `int`(), `float`(), `bool`(), `str`()
  - par exemple `print("J'ai " + str(31) + " ans"`
- Quelques autres :
  - `type`(), `exit`(), `print`(), ...
  - La liste complète se trouve dans le doc Python : `https://docs.python.org/3/library/functions.htm`

- Il y a assez peu de fonctions directement implémentées dans le langage
- Il en manque donc que l'on pourrait s'attendre à trouver
  - `cos()`, `sqrt()`, ...
- Beaucoup d'autres fonctions sont **disponibles via des modules**

Modules fréquemment utilisés

- `io` = input / outputs : lecture et écriture de fichiers
  - `open()`, `close()`
- random
  - Générer des nombres random.
  - `seed()`, `randrange()`, `randint()`, `random()`, `shuffle()`
- string
  - Manipulation de chaînes de caractères
  - `format()`

Module format

Pourquoi ?

- lisibilité, économie de code, flexibilité

```
1 # Formatage
2 valeur_s = "mot"
3 print("{}".format(valeur_s))
4 valeur_d = 3
5 print("{}".format(valeur_d))
6 valeur_f = 3.5
7 print("{}".format(valeur_f))
8 valeur_c = 47
```

Module format

```
1 # Mélanger les types
2
3 name = input("nom : ")
4 date = input("aujourd'hui, nous sommes le : ")
5 duree_du_cours = input("la durée du cours est de (en heures) : ")
6 print('Le {} octobre, {} aura passé {}h dans la salle J001.'.format(
      date,name,duree_du_cours))
```

Module format

fichier `demo_format.py` : lisibilité

```
1 print(lundi , " précède " , mardi , ", " , mercredi , ", " , jeudi , ",
      ", vendredi , ", ", samedi , ", et ", dimanche , ".")
2
3 print('{} précède {}, {}, {}, {}, {} et {}.'.format(lundi, mardi,
     mercredi, jeudi, vendredi, samedi, dimanche))
```

Module format

fichier `demo_format.py` : compacité

La liste de valeurs est parcourue grâce à l'opérateur $*$

```
1 jours = "Lundi mardi mercredi jeudi vendredi samedi dimanche"
2 liste_jours = jours.split()
3 print('{} précède {}, {}, {}, {}, {} et {}.'.format(*liste_jours))
```

Modules fréquemment utilisés

Fonctions système

- sys
  - Fonctions système
  - `argv()`, `getsizeof()`, `path()`, `stdin()`, `stdout()`, `stderr()`
- os
  - `chdir()`, `getcwd()`, `fchmod()`, `kill()`
  - messages d'erreur système `EX_OK` (status 0), `EX_OSERR`, etc.

Créer ses propres modules

- Pour importer un module *depuis la console de l'interpréteur python* il faut que celui-ci se trouve dans le même dossier que le module que l'ont veut importer
- Mettre une **docstring** permet d'ajouter du contenu à ce qui sera affiché par la fonction help().

docstring : chaîne de caractères avec trois doubles quotes """ situées au début d'un module, d'une fonction, d'une classe, d'une méthode.

Différences modules / scripts

- Scripts :
  - Se comportent comme des applications
  - Se lancent *via* la commande python3
- Modules :
  - Fournissent des fonctions et des variables aux scripts
  - Ne sont (généralement) pas exécutables
- Ce sont tous des fichiers source, la différence est dans leur utilisation.

exemple :
`https://github.com/python/cpython/blob/3.10/Lib/datetime.py`

from et import

L'import de modules peut également se faire de la manière suivante :

```
1 from module import fonction
2 from module import *
```

Lorsqu'on importe un module, la totalité de ce qui est exécutable est exécuté

Mot clé as

le mot-clef **as** permet de renommer localement (alias) un module importé

Scope des modules

- Les modules n'ont accès qu'aux variables globales de leur propre fichier.
- Le script a accès aux variables globales des modules qu'il a importé.

Scope main

- De nombreux langages de programmation ont une fonction spéciale qui est exécutée automatiquement en premier lorsque le programme est lancé.
- Lorsqu'un programme python est exécuté, l'interpréteur chercher un mot-clef particulier à exécuter en premier : **__main__**

interagir avec l'interpréteur Python : module `sys`

- `argv` = liste des arguments
- `len(argv)` = nombre d'arguments

\+ Démo `affiche_parametres.py`

Python Package Index

Puisque tout le monde peut écrire ses propres modules il en existe une grande quantité.

- Beaucoup peuvent être installée depuis PyPI
  - `https://pypi.org/`
- L'outil `pip3` est fourni avec Python est permet d'installer des paquets depuis PyPI.
  - `https://packaging.python.org/tutorials/installing-packages/`

Sources

- Cours de Pablo Rauzy (lien)
- Cours de Jean-Pascal Palus (lien)