



Méthodologie de la Programmation

Palus Jean-Pascal

Licence IV - L1 - Semestre 1

Bureau A184 - jpp@up8.edu

Séance I

Programmation impérative

- ▶ Python est un langage impératif.
- ▶ Dans un langage impératif, un programme est une suite d'instructions.
- ▶ Une instruction correspond à une “commande”, telle que :

- ▶ Python est un langage impératif.
- ▶ Dans un langage impératif, un programme est une suite d'instructions.
- ▶ Une instruction correspond à une “commande”, telle que :
 - La déclaration d'une variable
 - L'affectation de la valeur d'une expression à une variable
 - Un test conditionnel
 - Une itération

- ▶ Une variable dans un programme a le même rôle qu'une variable en mathématique : elle symbolise (nomme) une valeur qui peut changer au cours du temps.
- ▶ Une variable a un nom et un type.
- ▶ Le type définit les valeurs possibles : un nombre, une chaîne de caractères, un booléen...

- ▶ Une *variable* dans un programme a le même rôle qu'une variable en mathématique : elle symbolise (nomme) une valeur qui peut changer au cours du temps.
- ▶ Une variable a un nom et un type.
- ▶ Le type définit les valeurs possibles : un nombre, une chaîne de caractères, un booléen...
 - Dans certains langages le type est attaché à la variable et ne peut pas changer, on parle alors de langages *statiquement typés*.
 - Dans d'autres, le type est seulement attaché aux valeurs et donc le type d'une variable peut changer, on parle alors de langages *dynamiquement typés*.

- ▶ Une *expression* est une combinaison d'éléments du langage qui retourne une valeur quand elle est *évaluée* :

```
2 + 3
```

```
a > 20
```

```
taille * 100
```

```
prenom + " " + nom
```

- ▶ Dans certain langage, on doit déclarer une variable avant de pouvoir l'utiliser :
 - `var variable`
 - `int variable`
 - `bool variable`
 - ▶ Dans d'autres, il suffit d'affecter une valeur à la variable :
 - `nbr ← 42`
 - `titre ← "test"`
- ! Attention, dans certains langage l'affectation se fait avec le symbole `=`, sans que celui-ci n'ait de rapport avec le `=` des mathématiques !

- ▶ Un programme doit pouvoir faire des choix dynamiquement, lors de son exécution.
- ▶ On utilise pour cela des *tests conditionnels*.
- ▶ Ils permettent de n'exécuter un *bloc* (sous partie) du programme que si une expression booléenne est vraie :
 - si** $\text{age} \geq 18$ **alors** :
 afficher("majeur")
 - sinon** :
 afficher("mineur")

- ▶ Un programme doit pouvoir répéter certaines opérations plusieurs fois.
- ▶ On utilise pour cela des *boucles*.
- ▶ Elles permettent de d'exécuter un bloc du programme que tant qu'une expression booléenne est vraie :

$n \leftarrow 0$

tant que $n < 10$ **faire** :

 afficher_entier(n)

$n \leftarrow n + 1$

- ▶ 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
 renvoyer resultat

- ▶ Pour organiser les programmes, on utilise des *structures de données*.
- ▶ Une structure de données est un *type* qui regroupe plusieurs variables.
- ▶ Il y a différent type de structures de données.
- ▶ Selon les langages, certains types de structures de données existent nativement : les listes, les vecteurs, les dictionnaires..
- ▶ On peut aussi créer ses propres structures de données
 - Par exemple, une structure représentant un élève comprendra son prénom, son nom, son numéro d'étudiant·e, son adresse email, son UFR, son année d'étude, ses options, ...
 - En créant ainsi un type "élève" on peut utiliser une seule variable de ce type là où on a besoin de toutes ces valeurs.

Python

- ▶ Python a été créé en 1991 par Guido van Rossum.
- ▶ C'est un langage très répandu pour lequel il existe énormément de *bibliothèques*.
- ▶ Python est *dynamiquement typé*.
- ▶ Python est plutôt prévu pour la *programmation impérative*.
- ▶ Python offre un support de la programmation *orientée objet* (à base de classe).

- ▶ La *syntaxe* de Python se veut très lisible.
- ▶ L'indentation et les retours à la ligne sont significatifs :
 - une instruction se termine avec un retour à la ligne
 - les blocs sont marqués par l'indentation
- ▶ Les *commentaires* sont tout ce qui suit le symbole # sur une ligne jusqu'à la fin de celle-ci

- ▶ Les variable n'ont pas de types et il n'y a pas besoin de les déclarer.
 - ▶ La convention en Python est de nommer les variables en minuscules en séparant les mots par des underscores.
 - ▶ L'opérateur d'affectation est `=`.
 - ▶ Exemples :
 - `distance = speed * duration`
 - `length_in_cm = 2.54 * length_in_inch`
- ! Attention, ce `=` est différent du `=` des mathématiques !

- ▶ Les valeurs booléennes en Python sont **True** et **False**.
- ▶ Il existe aussi une valeur **None** qui veut dire “rien”.
- ▶ Les opérateurs de comparaison booléenne renvoient **True** ou **False**.
- ▶ Pour tester si une valeur `expr` est `None` on utilise `expr is None`.

- ▶ `==` : égal
- ▶ `!=` : différent
- ▶ `>` : strictement supérieur
- ▶ `>=` : supérieur ou égal
- ▶ `<` : strictement inférieur
- ▶ `<=` : inférieur ou égal

- ▶ Les nombres peuvent être entier (\mathbb{Z}) ou flottant ($\sim \mathbb{R}$).
- ▶ Exemples
 - 13.51
 - 42

- ▶ Les chaînes de caractères sont notées entre simple ou double quote (' ou ").
- ▶ Selon lequel on utilise il faut l'échapper avec un \.
- ▶ On peut utiliser certains opérateurs sur les chaînes comme * et +.

- ▶ Python permet de manipuler des paires, des triplets, des quadruplets, ...
- ▶ La syntaxe est de séparer les expressions par des virgules.
- ! Attention **1,23** est la paire composée de **1** et **23**, pas le nombre **1.23**.

- ▶ Une *liste* (ou *tableau*/*vecteur*, c'est confondu en Python) se notent entre crochets et leurs éléments séparés par des virgules.
- ▶ On accède à un élément d'une liste en donnant son indice entre crochets.
- ▶ Exemples :
 - `one_to_ten = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
 - `one_to_ten[3] # 4`

- ▶ Un *dictionnaire* (ou *tableau associatif*) en Python permet de stocker des associations clef-valeur.
- ▶ On note les dictionnaires entre accolades, leurs entrées séparées par des virgules, et les clefs séparées des valeurs par des :
- ▶ On accède à une valeur avec sa clef entre crochets.

- ▶ La syntaxe des conditions est la suivante :
 - if** condition :
 - then-block
 - elif** other-condition :
 - otherwise-block
 - else** :
 - else-block
- ▶ Il peut y avoir zéro ou plusieurs bloc **elif** après un bloc **if**.
- ▶ Il peut y avoir zéro ou un bloc **else** à la fin.
- ▶ Les branches sont introduites par un symbole : puis délimitées par l'indentation.
- ▶ (La convention en Python est d'utiliser 4 espaces comme indentation)

- ▶ Il existe deux types de boucles :
 - “tant que”, qui répète un bloc d’instructions tant qu’une condition est vraie.
 - “pour ... dans”, qui répète un bloc d’instructions pour chaque valeur dans un conteneur.
- ▶ Leur syntaxe sont :
 - **while** bool-expr :
do-block
 - **for** v **in** iterable :
do-block
- ▶ Même syntaxe que pour les branches conditionnelles.

- ▶ Il y a une syntaxe spéciale pour faire des opérations sur les éléments d'une liste.
 - `[expr(x) for x in lst]`
- ▶ Pour créer une nouvelle liste à partir des éléments d'une liste existante :
 - `[expr(x) for x in lst if pred(x)]`

- ▶ Les *fonctions* en Python sont définies avec le mot clef `def` de la manière suivante :
 - **def** *function_name* :
- ▶ La convention en Python est de nommer les fonctions en minuscules en séparant les mots par des underscores.
- ▶ On utilise **return** expr pour renvoyer une valeur et quitter la fonction.
 - Utilisé sans rien derrière est équivalent à **return** None.
- ▶ Appels de fonctions :
 - `fact(10)`
 - `distance = get_distance(50, 0.25)`

- ▶ On appelle la *portée* (ou *scope*) d'une variable la 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 dans une fonction n'est pas une modification "en place", elle crée une variable locale.

- ▶ En Python les arguments sont passés aux fonctions par *référence*.
- ▶ Cela veut dire que si on fait une modification *en place* de ces variables, alors elles seront affectées en dehors de la fonction aussi.

- ▶ Il est possible de spécifier une valeur par défaut pour certains arguments (qui doivent tous être après ceux qui n'ont pas de valeur par défaut).

- ▶ Écrire un programme qui affiche les nombres de 0 à 100, mais remplace les nombres divisibles par 3 par “Trois”, ceux divisibles par 5 par “Cinq”, et ceux qui le sont par 3 et par 5 par “Trois et Cinq”.
- ▶ Ecrire une fonction retournant la liste des n premiers nombres de Fibonacci sachant que :
 - $F_0 = 0$
 - $F_1 = 1$
 - $F_n = F_{n-1} + F_{n-2}$