



INTRODUCTION À LA PROGRAMMATION EN PYTHON

Organisation du cours:

Environ 30 heures de cours/TD

Projet individuel ou en binôme: sujet libre sous réserve de validation

Exemple: implémenter et résoudre un labyrinthe, coder un algorithme de graph, implémenter un réseau de neurones...



Évaluation:

Projet obligatoire, au minimum 50% de la note finale.

Au choix:

- Un rendu par semaine
- Un exo de 5 minutes en début de chaque cours
- Un DS sur ordinateur d'une ou deux heures



Sommaires

1. Rappels sur un ordinateur
2. Premiers pas en python: variable, types de données et *if*
3. Les conteneurs de données
4. Les boucles *for* et *while*
- 5.



Introduction/Rappels sur un ordinateur

Principe de l'ordinateur

« Automate déterministe à composants électroniques »

Trois parties principales:

- processeur: exécute des instructions
- mémoire vive: mémoire rapide d'accès, mais éphémère où sont chargés les programmes en cours d'exécution
- disque dur: mémoire plus lente, qui n'est pas effacée lorsque l'ordinateur s'éteint



La Programmation

Un langage de programmation fait le lien entre l'humain (le langage naturel) et l'ordinateur (le binaire).

Un algorithme est une description précise des opérations à faire pour arriver à un résultat.

Les programmes sont la traduction dans un langage de programmation d'un algorithme.



Pourquoi Python?

- Haut niveau, i.e. il admet une certaine abstraction.
- Portable, i.e. peut être utilisé sur toute les systèmes d'exploitation
- Libre et gratuit
- Très utilisé, donc il y a une très bonne communauté sur internet
- De très nombreux modules sont disponibles gratuitement



Comment exécuter du code Python ?

Trois façons:

- L'interpréteur ou Shell
- Les scripts et modules:
- Les Jupyter Notebook:

>>> Instruction 1

Résultat 1

>>> Instruction 2

Résultat 2

#!/usr/bin/python3

NomDuScript

Instruction 1

Instruction 2

Description en Markdown

>>> Serie d'instruction 1

Résultat 1

>>> Serie d'instruction 2

Résultat 2



Récapitulatif

- Les instructions sont exécutées les unes à la suite des autres
- Le projet final devra être sous forme de script, mais le rapport peut être un jupyter Notebook
- Nous travaillerons sur des Jupyter Notebook

Premiers pas en Python

Les variables et les types de données

Pour manipuler des objets, il faut utiliser les variables. Une variable est un nom que l'on donne à un objet.

a = 1 signifie que l'on peut accéder au nombre 1 en appelant la variable *a*.

a = *b* = 1 signifie que le nombre 1 a 2 étiquettes.

Précisions: si on entre les instructions

a = 1

b = *a*

a = 10

La variable *b* contient encore 1.

Pour voir ce que contient une variable, il faut utiliser la fonction *print*.

Exemple:
print(a) renvoie ici 10.

Les variables et les types de données

Les variables peuvent contenir plusieurs types de données:

- des entiers (`int`): 1, 2, 165464564536, ...
- des réels (`float`): 3.1418, 0.5, 1., ...
- des booléen (`bool`): True, False
- des caractères/chaînes de caractères (`str`): 'a', '0', '#', 'prénom', 'Ceci est une chaîne de caractères. 15264, True, 0.3564'...

Pour accéder au type d'une variable, il existe une fonction *type*.

type(1) renvoie *int*

Les variables et les types de données

On peut forcer le type d'une données:

- `int('5')` vaut 5, `int(4.5)` vaut 4, `int(True)` vaut 1, `int(False)` vaut 0
- `float(3)` vaut 3., `float('2.5')` vaut 2.5, `float(True)` vaut 1., `float(False)` vaut 0.
- `str(True)` vaut 'True', `str(2)` vaut 2
- `bool(x)` vaut `x != 0` si x est un entier
- `bool(s)` vaut `s != ""` (chaînes de caractères vide) si s est une string

`!=` signifie différent. Pour tester l'égalité de deux variables, on utilise `==`. Plus grand ou égal est symbolisé par `>=`, et `<=` pour plus petit ou égal



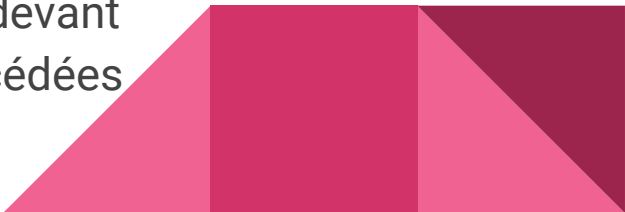
Les instructions conditionnelles (1/3)

Certaines instructions ne doivent être exécutées que sous certaines conditions. Par exemple, dans un jeu, une ligne de code peut afficher *Gagné !* uniquement quand le joueur a gagné.

Pour cela, on utilise la syntaxe:

if condition:
 instructions

Attention, les deux points après la condition et l'indentation devant les instructions sont importants. Seules les instructions précédées d'une indentation seront soumises à la condition.



Les instructions conditionnelles (2/3)

Dans certains cas, on peut vouloir exécuter une instruction si une condition est remplie et une autre si cette condition n'est pas remplie. Dans ce cas, la syntaxe est la suivante:

if condition:

instructions si la condition est remplie

else:

instructions si la condition n'est pas remplie



Les instructions conditionnelles (3/3)

Enfin, il est possible de vouloir d'exécuter une instruction si une première condition est remplie, une autre instruction si la première condition n'est pas remplie mais une seconde l'est, et encore une autre instruction si les deux conditions ne sont pas remplies.

if condition1:

instruction exécutée si condition1 est remplie

elif condition2:

instruction exécutée si condition1 n'est pas remplie et condition2 l'est

else:

instruction exécutée si condition1 et condition2 ne sont pas remplies



Syntaxe des conditions

Derrière un if, il faut mettre une expression booléenne.

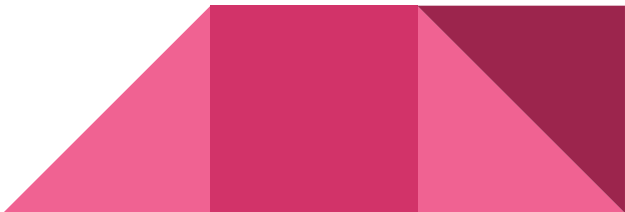
Exemple:

- `if x == 10:`
- `if x > 2:`
- `if s == "Alexandre":`
- `if x>2 and y>2:`
- `if b:`

Ici, x et y sont des entiers, s est une chaîne de caractères et b est un booléen.

Pour tester l'égalité d'une variable, on utilise un double égal `==` (= étant utilisé pour attribuer une valeur à une variable).

Attention, écrire `b == True` est correct, mais inutile, car b est déjà un booléen.



Récapitulatif

- Les données sont stockées dans des variables
- Les données sont typées
- Les instructions conditionnelles se font grâce aux mots clefs if, elif et else
- Après ces mots clefs, il faut :
puis une indentation avant les lignes suivantes

Les conteneurs de données

Des types de données plus complexes

Ils existent des types de données qui sont des conteneurs:

- les listes: [valeur1, valeur2, valeur3, ...]
- les sets: { valeur1, valeur2, valeur3}
- les dictionnaires ou maps: {clef1: valeur1, clef2: valeur2}



Les listes

Les listes sont entre crochets en python. `[]` est la liste vide. Les éléments sont séparés par des virgules. On utilise les listes pour stocker des données en conservant l'ordre.

Quelques opérations classiques, avec `l = [2, 5, 1, 9]` :

- `+` : concaténation. `l + [3, 4]` renvoie `[2, 5, 1, 9, 3, 4]`
- `append` : ajout d'un élément. `l.append(3)` attribue à `l` la valeur `[2, 5, 1, 9, 3]`
- `[]` : accès à un élément. `l[2]` renvoie `1`.
- `len` : la longueur de la liste. `len(l)` renvoie `4`.

Attention, on indexe à partir de 0. `l[0] = 2`



Les sets

Les sets sont entre { } en python. `set()` est le set vide. Les éléments sont séparés par des virgules. On utilise les sets pour stocker des éléments de manière unique, mais désordonnée.

Quelques opérations classiques, avec `s = {2, 5, 1, 9}` :

- **union** : concaténation. `s.union({3, 4, 5})` renvoie `{2, 5, 3, 4, 9, 1}`
- **add**: ajoute un élément. `s.add(3)` attribue à `s` la valeur `{2, 5, 1, 9, 3}`
- **remove**: supprime un élément. `s.remove(2*)` attribue à `s` la valeur `{5, 1, 9}`.
- **len**: la taille du set. `len(s)` renvoie 4.
- **in**: test si un élément est dans un set. `5 in s` renvoie `True`.

Listes et sets

- `in` fonctionne pour les deux, mais est plus efficace avec les sets.
- Les listes conservent l'ordre, pas les sets.
 - => "le troisième élément d'un set" n'a pas de sens
 - => on peut trier une liste, pas un set
- Pour les listes comme les sets (et les dictionnaires), si on exécute:

```
l = [1, 2, 3]
```

```
m = l
```

```
l.append(4)
```

`m` a alors la valeur `[1, 2, 3, 4]`, mais

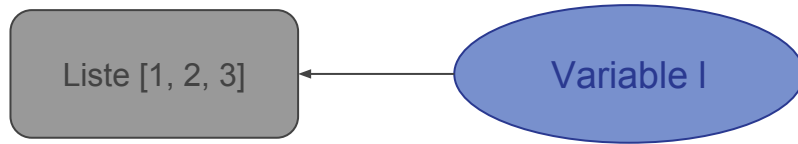
```
l = [5, 6, 7]
```

ne change pas la valeur de `m`.



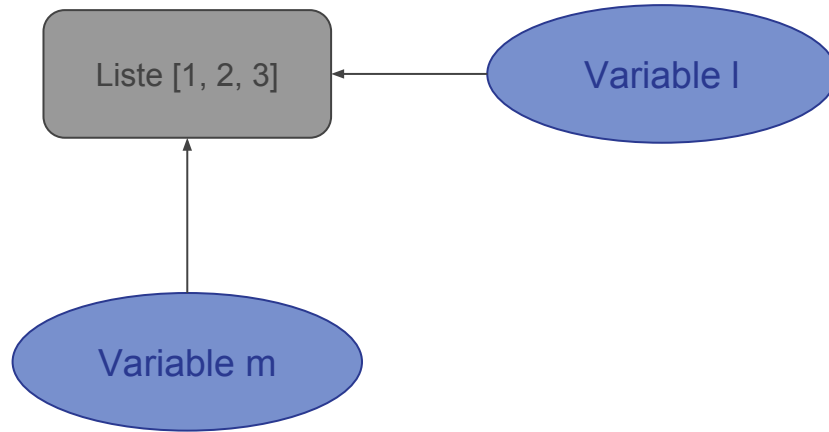
Explications

$I = [1, 2, 3]$



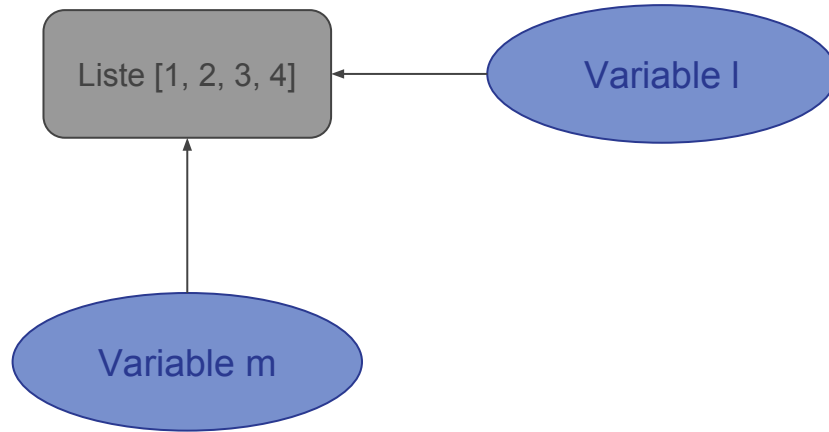
Explications

$m = l$



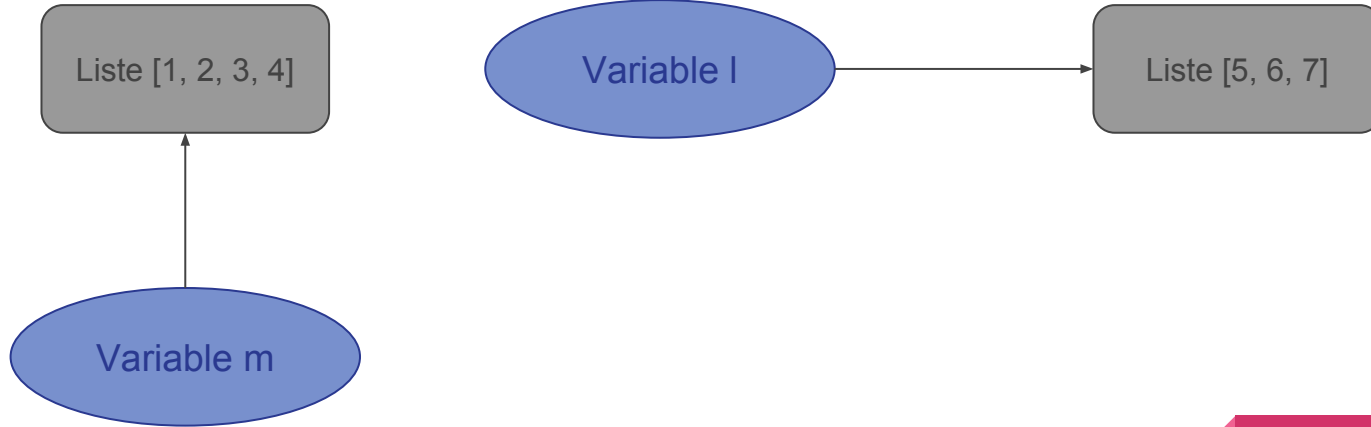
Explications

`l.append(4)`



Explications

$I = [5, 6, 7]$



Fonction inplace

Une fonction inplace est une fonction qui change l'objet sur lequel elle est appelée. Exemple: append, add, remove, sort

Pour éviter ce problème, il faut faire une copie de l'objet. Pour ce faire, il faut appeler le constructeur du type associé.

Exemple:

```
l = [1, 2, 3]
```

```
m = list(l)
```

```
l.append(4)
```

```
m contient alors [1, 2, 3]
```

Une définition précise de fonction et de constructeur est donné plus loin.



Dictionnaire

Les dictionnaires sont aussi entre { } en python. {} est le set vide. Les éléments sont séparés par des virgules. On utilise les dictionnaires pour stocker des paires (clef, valeur). Une clef et une valeur sont séparées par deux points.

Quelques opérations classiques, avec `d = {'c1': 'v1', 'c2': 'v2'}` :

- `[]`: renvoie la valeur associé à une clef. `d['c1']` renvoie 'v1'.
- `get`: cherche une clef dans un dictionnaire, renvoie sa valeur si elle est présente et une valeur par défaut sinon. `d.get('c3', 'v3')` renvoie 'v3'.
- `len`: le nombre de paires dans le dictionnaire. `len(d)` renvoie 2.
- `in`: test si une clef est dans un dictionnaire.
`'c1' in d` renvoie `True`.

Récapitulatif

- les listes sont entre [] et conservent l'ordre des éléments
 - les sets sont entre { } et stockent des objets uniques
 - les dictionnaires sont aussi entre { } et stockent des paires (clef, valeur)
 - Attention aux fonctions changeant les objets sur lesquels elles sont appelées!
-

Les boucles

