Initiation à Python

Séance 1

Valentin Bahier

2020-2021

Plan

Plan

Plan

Motivations

- Nombreux domaines d'utilisation (marketing, communication, finance, big data, intelligence artificielle, jeux, web...)
- Langage facile à apprendre
- Grande communauté
- Connaître les bases de codage peut ouvrir des portes

Objectifs du module

- S'initier à la programmation avec Python
- Découvrir les concepts fondamentaux de programmation
- Comprendre la structure et le fonctionnement d'un programme informatique simple
- Mener un mini-projet de programmation

⚠ Frustration de "débuggage"

Plan

Planification

Semaine 1

- Premiers pas avec Python, structures de données, conditions
- Boucles, fonctions, méthodes
- Modules `random` , `numpy` , `turtle`
- Représentations graphiques avec `matplotlib`
- Une étude de cas

Semaine 2

- Réalisation d'un mini-projet par petits groupes
- Approfondissements personnels

Modalités d'évaluation

Evaluations

- QCM réguliers à chaque séance (5 points)
- Mini-projet en trinôme (5 points)
- Test final (10 points)

Plan

Plan

D'où vient Python ?

Guido Van Rossum

Création en 1989 par Guido Van Rossum (Néerlandais né en 1956)

Il est fan des *Monty Python*.

Plan

Python 2 ? Python 3 ?

Actuellement, Python possède deux grandes versions : la version 2, et la version 3.

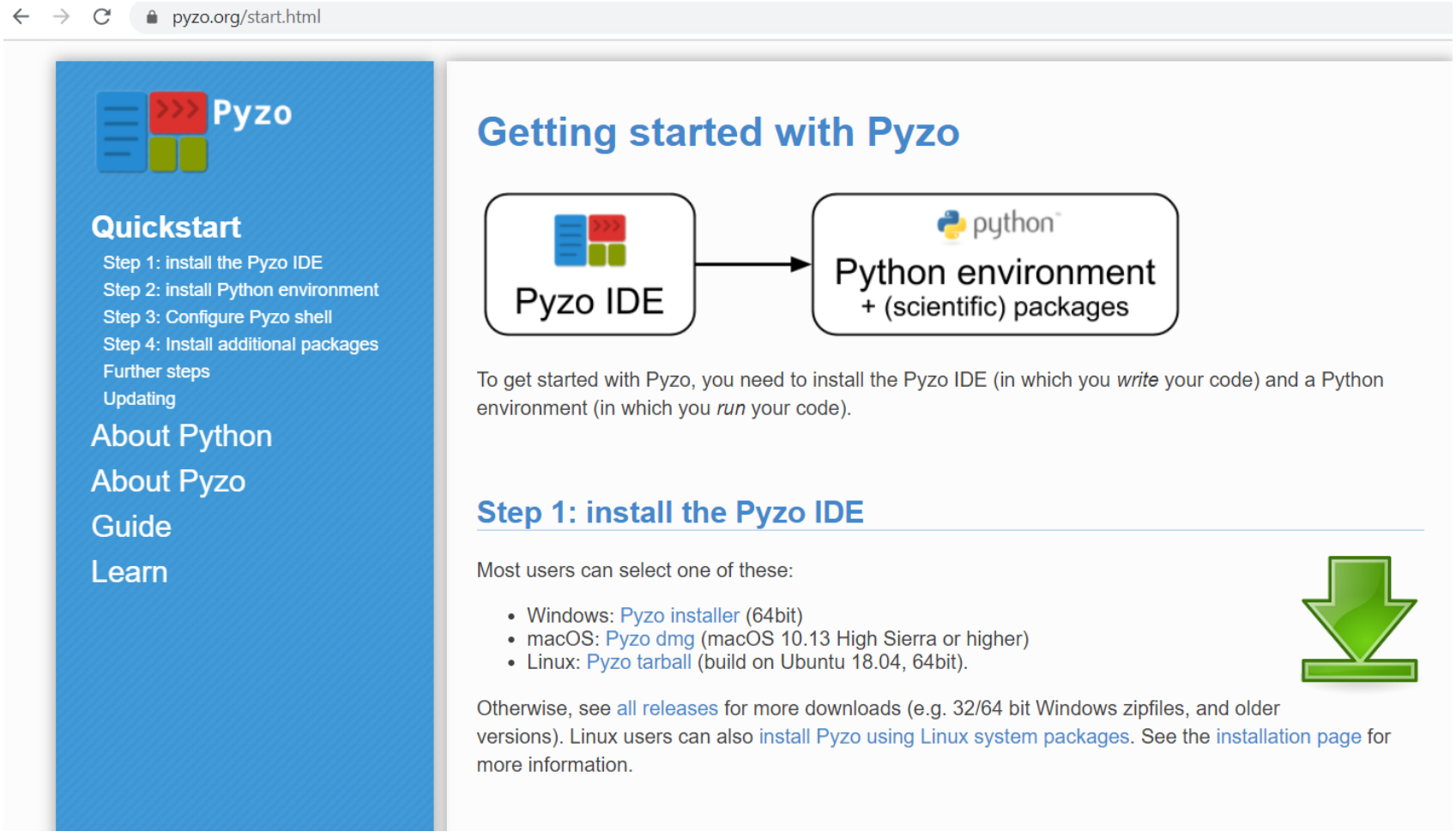
⚠ La version 3 n'est pas rétrocompatible avec la version 2.

Depuis le 1er janvier 2020 la version 2 commence à poser quelques problèmes, dont des problèmes de sécurité (car il a été décidé officiellement qu'à partir de cette date cette version ne sera plus mise à jour).

Nous choisirons donc la version 3 pour ce cours.

Choisir un bon environnement de travail

Il existe de nombreux bons environnements de développement pour programmer avec Python. Pour ce cours, nous travaillerons avec **Pyzo**.



Plan

Python comme calculatrice

Les quatre opérations de base (addition, soustraction, multiplication, division) se font avec les symboles `+`, `-`, `*`, `/`. Quelques autres opérations possibles :

`x**y`	x^y
`a%b`	reste de la division euclidienne de a par b
`a//b`	quotient de la division euclidienne de a par b

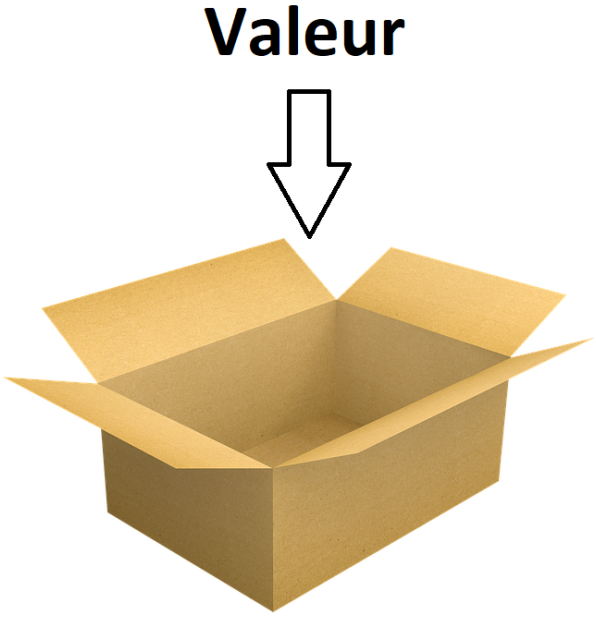
En particulier si on souhaite obtenir la racine carrée d'un nombre x il suffit de taper

```
x**0.5
```

⚠ Les nombres à virgule s'écrivent avec un point au lieu de la virgule (notation anglo-saxonne). On écrira par exemple `2.8` au lieu de `2,8`.

Plan

Une variable, de quoi s'agit-il ?



Variable

Ordinateur

En Python, l'affectation s'effectue grâce au symbole =.

```
ma_variable = valeur
```

Quels noms donner aux variables ?

Toutes les lettres de l'alphabet (minuscules et majuscules) sont autorisées. Il n'est en revanche pas permis d'utiliser d'accent, d'espace, ou de caractères spéciaux (hormis le symbole _).

Il est d'usage de toujours commencer les noms de variables par une minuscule, puis au choix :

- soit mettre une majuscule pour chaque nouveau mot.

```
maVariableExemple
```

- soit mettre un underscore entre chaque mot mais sans aucune majuscule.

```
ma_variable_exemple
```

Exercice 1

- Qu'affiche la console de Python après avoir donné les instructions suivantes ?

```
1 a = 2
2 a = a+1
3 b = 5.6
4 c = a*b
5 print(c)
```

- Maintenant, fermer l'environnement de travail puis le rouvrir, et entrer l'instruction

```
1 print(c)
```

Que se passe t-il ?

Exercice 1

- Qu'affiche la console de Python après avoir donné les instructions suivantes ?

```
1 a = 2
2 a = a+1
3 b = 5.6
4 c = a*b
5 print(c)
```

- Maintenant, fermer l'environnement de travail puis le rouvrir, et entrer l'instruction

```
1 print(c)
```

Que se passe t-il ?

Réponse : La console affiche `16.8`. Après avoir fermé puis rouvert, la console renvoie une erreur.

$\Rightarrow$ Mémoire courte. Les valeurs des variables sont seulement stockées le temps d'une session. Cela donne beaucoup d'intérêt aux **scripts**, qui eux en revanche vont pouvoir être sauvegardés.

Types de variables

Les valeurs des variables peuvent être des nombres mais aussi d'autres données (du texte, des listes, des tableaux...). La nature de la donnée qui se trouve dans une variable constitue le **type** de cette variable (ou le type de la donnée). Les méthodes et les fonctions que l'on va pouvoir appliquer aux variables diffèrent suivant leur type.

Parmi les types les plus répandus on trouve

- `int` (comme *integer* en anglais) : nombre entier
- `float` : nombre flottant (nombre décimal)
- `str` (comme *string* en anglais) : chaîne de caractères
- `list`, `tuple`, `dict`, `bool`...

Exercice 2

Rentrer les instructions suivantes dans la console

```
1 a = 7
2 b = 4.
3 c = "oui"
4 type(a)
5 type(b)
6 type(a+b)
7 type(c)
8 print(c+c)
9 type(c+c)
```

Commenter ce qu'il se passe pour chaque ligne.

Exercice 2

Rentrer les instructions suivantes dans la console

```
1 a = 7
2 b = 4.
3 c = "oui"
4 type(a)
5 type(b)
6 type(a+b)
7 type(c)
8 print(c+c)
9 type(c+c)
```

Commenter ce qu'il se passe pour chaque ligne.

Réponse : Le type de `a` est `int`, celui de `b` et `a+b` est `float`, celui de `c` et `c+c` est `str`. L'addition de deux chaînes de caractères est la **concaténation**.

$\Rightarrow$ Le symbole `+` a donc un sens différent suivant les types de données entre lesquels il intervient.

Plan

Plan

Les listes

Une **liste** est une collection d'éléments (pas nécessairement de même type) séparés par des virgules, l'ensemble étant enfermé dans des crochets.

```
L = []                                                    liste vide
fruits = ["abricot", "banane"]      liste contenant les deux chaînes de caractères "abricot" et "banane"
```

Exercice 3

Rentrer les instructions suivantes dans la console

```
L = ["a","b","c","d","e"]
L[1]
L[0]
L[3]
L[-1]
L[2:4]
L[:3]
L[3:]
len(L)
```

Deviner ce que renvoie chaque instruction.

Exercice 3

Rentrer les instructions suivantes dans la console

```
L = ["a","b","c","d","e"]
L[1]
L[0]
L[3]
L[-1]
L[2:4]
L[:3]
L[3:]
len(L)
```

Deviner ce que renvoie chaque instruction.

Réponse : `L[k]` est le $(k+1)$-ième élément de la liste L. `L[-1]` est son dernier élément. `L[i:j]` est la liste de tous les éléments entre le $(i+1)$-ième et le j-ième (inclus). `L[:j]` est la liste de tous les éléments jusqu'au j-ième (inclus), et de même `L[i:]` est la liste de tous les éléments à partir du $(i+1)$-ième. Enfin, `len(L)` est la longueur de L (c'est-à-dire son nombre d'éléments).

$\Rightarrow$ Le premier indice d'une liste est 0 et non 1 !

Plan

Les tuples

Comme les listes, les **tuples** sont des collections ordonnées d'éléments. La différence majeure est qu'une fois définis, les tuples ne peuvent pas être modifiés. On dit qu'ils sont *non mutables*. On met des parenthèses au lieu des crochets.

```
T = ()                                                    tuple vide
coordonnees = (2,0,5) tuple contenant les nombres entiers 2, 0
et 5
```

Exercice 4

Rentrer les instructions suivantes dans la console et observer ce qui s'affiche

```
1 L = ["A","b"]
2 T = ("A","b")
3 T[0]
4 T[1]
5 L[1] = "B"
6 L
7 T[1] = "B"
```

Exercice 4

Rentrer les instructions suivantes dans la console et observer ce qui s'affiche

```
1 L = ["A","b"]
2 T = ("A","b")
3 T[0]
4 T[1]
5 L[1] = "B"
6 L
7 T[1] = "B"
```

Réponse : Tout se passe bien sauf pour la dernière instruction où Python renvoie une erreur.

$\Rightarrow$ On peut changer la valeur d'un élément d'une liste, mais pas d'un tuple.

Plan

Les dictionnaires

Les **dictionnaires** rassemblent eux aussi des éléments, mais à la différence des listes et tuples où chaque élément est repéré par un indice, chaque élément d'un dictionnaire est repéré par une **clé**.
On utilise des accolades pour les dictionnaires (et non des crochets ou parenthèses).

```
D = {}                                                    dictionnaire vide
effectifs = {"hommes": 20, "femmes": 16}      dictionnaire
contenant les deux nombres entiers 20 et 16 respectivement
associés aux clés "hommes" et "femmes"
```

Exercice 5

Rentrer les instructions suivantes dans la console et observer ce qui s'affiche

```
1 mon_frigo = {"yaourts": 8, "legumes": ["tomate", "
      aubergine"]}
2 mon_frigo["yaourts"]
3 mon_frigo["legumes"]
4 mon_frigo["legumes"][1]
5 mon_frigo["yaourts"] = 7
6 mon_frigo
```

Exercice 5

Rentrer les instructions suivantes dans la console et observer ce qui s'affiche

```
1 mon_frigo = {"yaourts": 8, "legumes": ["tomate", "
    aubergine"]}
2 mon_frigo["yaourts"]
3 mon_frigo["legumes"]
4 mon_frigo["legumes"][1]
5 mon_frigo["yaourts"] = 7
6 mon_frigo
```

Réponse : `mon_frigo["yaourts"]` permet d'accéder à la valeur associée à la clé `"yaourts"`. `mon_frigo["legumes"]` est une liste, donc on accède à la valeur `"aubergine"` en tapant `mon_frigo["legumes"][1]`.

$\Rightarrow$ Les listes, tuples, dictionnaires peuvent se combiner entre eux autant de fois que l'on souhaite.

Plan

Toujours commencer avec du pseudo-code

Avant de se lancer dans l'écriture d'un programme qui va exiger plus d'une dizaine de lignes de code, il est vivement conseillé de dresser une esquisse de code avec ce qu'on appelle du **pseudo-code**, mettant au clair

- les principales étapes du programme
- les structures de données choisies

Le pseudo-code consiste en un ensemble d'instructions en "français presque naturel" décrivant le programme.
Exemple :

```
Si couleur_petit_bonhomme = "vert"
  alors traverser_la_rue
Sinon
  appuyer_sur_le_bouton
```

Commenter les scripts

Rappel sur l'intérêt du script : permet de modifier simplement des instructions, et de les enregistrer sur l'ordinateur.

Lorsqu'on travaille à plusieurs sur un même projet de programmation, ou bien lorsqu'on revient soi-même sur un vieux projet, tout devient plus simple lorsque les noms de variables sont limpides et que le code est **commenté** afin de vite se l'approprier.

Les commentaires en Python se font grâce au symbole #. **Tout ce qui suit un signe # sur une même ligne ne sera pas interprété par Python**. Il est également possible de faire un commentaire sur plusieurs lignes, en l'encadrant entre deux triples-guillemets """.

(avec Pyzo : $\boxed{\text{ctrl}}+\boxed{\text{R}}$ pour commenter, $\boxed{\text{ctrl}}+\boxed{\text{T}}$ pour décommenter)

Indentation

Python est un langage avec **indentation**. Cela signifie que certaines instructions doivent obligatoirement s'agencer de manière précise avec le bon nombre d'espaces devant chacune, sans quoi l'exécution du code renvoie des erreurs. On le verra plus précisément lorsque nous aborderons les *conditions*, les *boucles*, et les *fonctions*.

Pyzo (comme beaucoup d'autres environnements de développement Python) propose automatiquement l'indentation adéquate lors des retours à la ligne.

⚠ Gare aux fausses manipulations qui ajoutent ou suppriment les espaces requis devant certaines lignes.

Plan

Booléens

Un **booléen** est un type de donnée extrêmement simple qui prend seulement deux valeurs : `True` et `False`.

```
1 a = True
2 b = not(a)    # b prend la valeur contraire de a, donc
      vaut False
```

Opérateurs de comparaison

Un **opérateur de comparaison** permet, comme son nom l'indique, de comparer des valeurs. Par exemple, pour tester l'égalité entre deux nombres x et y, on tape `x==y`. Python renvoie alors un booléen (`True` si l'égalité est vraie, `False` sinon).

Les principaux opérateurs de comparaisons sont les suivants

`x==y`	x est égal à y
`x!=y`	x est différent de y
`x<y`	x est strictement inférieur à y
`x<=y`	x est inférieur ou égal à y

Exercice 6

Rentrer les instructions suivantes dans la console et commenter ce qu'elles font

```
1 x = 2
2 a = x==5
3 b = x<5
4 c = a or b
5 d = a and b
6 c,d
7 type((c,d))
```

Exercice 6

Rentrer les instructions suivantes dans la console et commenter ce qu'elles font

```
1 x = 2
2 a = x==5
3 b = x<5
4 c = a or b
5 d = a and b
6 c,d
7 type((c,d))
```

Réponse : a, b, c et d sont des booléens valant respectivement False, True, True et False. En effet, or est l'opération logique "ou", et and est l'opération logique "et". A la ligne 6, l'écriture désigne un tuple (il est permis d'enlever les parenthèses), on vient d'économiser une ligne de code !

⇒ On peut se servir des opérations logiques and , or , et not sur les booléens.

Plan

Les conditions

Bien souvent on est amené à vouloir donner des instructions uniquement lorsque certaines conditions sont satisfaites. C'est tout l'intérêt de l'instruction `if` , qui s'emploie de cette manière

```
if condition:
  bloc d'instructions 1
else:
  bloc d'instructions 2
```

⚠ Ne pas oublier les symboles `:`. Toutes les lignes d'un bloc d'instructions doivent être indentées de 2 (ou 4) espaces supplémentaires par rapport au `if` ou `else` .

Si on souhaite recourir à des conditions supplémentaires, on peut soit les imbriquer, soit utiliser un ou plusieurs `elif` entre `if` et `else` .

Exercice 7

- Créer un script appelé *condition.py* qui commence par l'instruction

```
1 ma_note = 12
```

puis qui affiche "admis" si `ma_note` est entre 10 et 20, et "rattrapage" si `ma_note` est en dessous de 10.

- Exécuter le script, puis changer la valeur de `ma_note` et réessayer.

Exercice 7

- Créer un script appelé *note.py* qui commence par l'instruction

```
1 ma_note = 12
```

puis qui affiche "admis" si `ma_note` est entre 10 et 20, et "rattrapage" si `ma_note` est en dessous de 10.

- Exécuter le script, puis changer la valeur de `ma_note` et réessayer.

Réponse :

```
1 ma_note = 12   #la note entre 0 et 20 que je vais
      avoir sans trop forcer
2 if ma_note>=10 and ma_note<=20:
3   print("admis")
4 elif ma_note<10:
5   print("rattrapage")
6 else:
7   print("impossible je ne peux pas etre si fort...")
```

$\Rightarrow$ Il y aura une fonction bien pratique (la fonction `input`) permettant de demander directement à l'utilisateur une valeur (pour donner à la variable `ma_note`) sans avoir à le faire manuellement. On verra cela à la prochaine séance.

Conclusion

Dans cette séance nous nous sommes initiés au langage Python grâce aux

- opérations calculatoires simples
- variables et types
- chaînes de caractères, listes, tuples, dictionnaires
- booléens et opérateurs de comparaison
- instructions conditionnelles

Dans la prochaine séance nous verrons des méthodes sur les différentes structures de données, et nous aborderons

- les boucles
- les fonctions