TP de Python numéro 1 - Prise en main

*Semaine du **14 septembre 2023**.*

Un premier TP de prise en main de `Python` avec l'IDE Pyzo. Quand du code `Python` est tapé sans consigne, vous devez le taper et l'exécuter pour voir ce qu'il fait.

I. Prise en main de Pyzo

1. Installer `Python` et Pyzo sur votre ordinateur personnel

Il est fortement conseillé d'installer `Python` et Pyzo sur votre ordinateur personnel. C'est très simple : rendez vous sur la page (in english)

`https://pyzo.org/start.html`

Follow steps 1 to 4 entirely. For step 2, take the first choice (regular python) and remember where you install Python (you might need it for step 3). For step 4, please type :

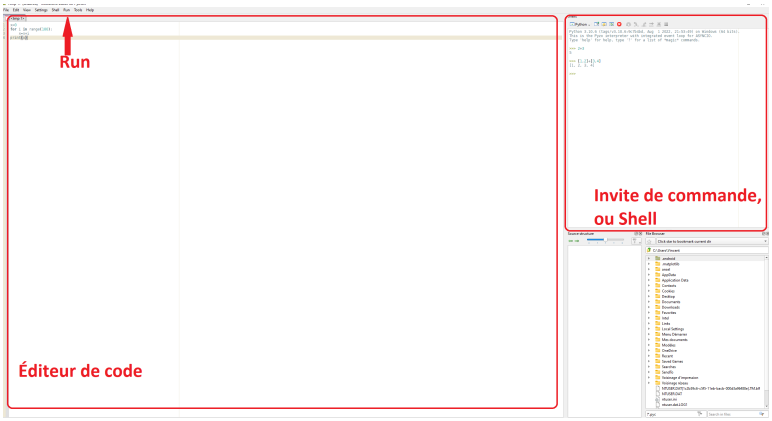
```
install numpy scipy pandas matplotlib sympy
```

(as we won't need the pyqt package).

Voilà, Pyzo et `Python` sont installés sur votre machine.

2. Découvrir l'interface de Pyzo

Pyzo est donc un *IDE* (environnement de développement intégré) pour `Python`, c'est à dire un éditeur de texte dans lequel vous allez taper du code, et qui est capable de communiquer avec `Python` pour exécuter vos programmes. Lancez Pyzo sur l'ordinateur.



Vous pouvez déjà identifier deux espaces : l'invite de commande et l'éditeur de code. **Dans la partie restante** de Pyzo, en bas à droite, vous avez un explorateur de fichiers et, à côté, une visualisation de l'environnement (vous comprendrez plus tard).

a) L'invite de commande

Dans **l'invite de commande**, toute ligne que vous tapez validée par `Entrée` est immédiatement interprétée par `Python`. C'est **très utile** pour vérifier des commandes et faire vos petits tests pendant que vous codez.

Remarque. Le triple chevron >>> indique que l'invite de commande est prête à recevoir une instruction.

Exercice 1. Dans l'invite de commande, tapez, validez et observez ces lignes. Puis commentez sur cette feuille :

```
>>> 25.2+1
>>> 25,2+1
>>> [2,3]+5
>>> x=10
```

```
>>> x=x+25
>>> print(x)
```

Remarque. Dans l'invite de commande, on a une affectation de variable globale : `Python` se rappelle de la ligne `x=10` à partir du moment ou elle est tapée. Le bouton *terminate and restart shell* permet de réinitialiser votre environnement, c'est à dire de tout désaffecter (entre autre).

b) L'éditeur de code

Dans **l'éditeur de code**, vous tapez comme sur un éditeur de texte classique (avec quelques fonctionnalités en plus). Si vous allez dans le menu `Run` et que vous sélectionnez la première option (`Run file as script` raccourci : Ctrl+Maj+E), le code tapé sera lu et interprété par l'invite de commande. Plus précisément, les lignes de l'éditeur de code seront lues et exécutées l'une après l'autre.

Remarque. Pour faire cela, il faut auparavant enregistrer le code tapé (dans l'éditeur de commande) dans un fichier (menu fichiers > enregistrer).

Exercice 2. Dans l'éditeur de code, tapez le code suivant et exécutez le. Commentez. S'il y a une erreur, retirez la ligne problématique et relancez.

```
print("Hello World")
x=10
x=x+25
print(x)
[2,3]+5
25,2+1
2 + 9   #les espaces sont optionnels
7+3*4
(7+3)*4 #La priorité des opérations est-elle respectée?
```

Remarque. Tout texte tapé après le symbole # n'est pas lu par la machine. Cela permet de commenter notre code.

Remarque. La fonction python `print()` affiche à l'écran ses arguments.

Exercice 3. Modifiez les lignes tapées ci-dessus de la manière suivante et exécutez. Expliquez.

```
print("Hello World")
x=10
x=x+25
print(x)
print(2 + 9)
print(7+3*4)
print((7+3)*4)
```

II. Prise en main de Python

Attention : Python est un langage sensible à la casse : il prends en compte la différence entre majuscule et minuscule.
Attention : Pour ne pas désespérer, vous devez apprendre à lire les erreurs renvoyées par Python (un minimum).

On peut dire qu'un programme informatique a vocation à transformer des données. Par exemple, on peut facilement concevoir un programme au cahier des charges suivant :

- Entrées : un entier n.
- Sortie : Affiche l'entier 0 si n est pair, l'entier 1 si n est impair.

Ce programme transforme la donnée d'entrée (n ici) en une nouvelle donnée (0 ou 1 ici), qui est ici juste affichée à l'écran. Voici une première manière de le faire (peu optimisée) :

```
n=42                          #l'entier donné en entrée est ici 42
if n%2==0:                       #si n est pair
        Resultat=0            #la variable Resultat prend la valeur 0
else:                       #sinon
        Resultat=1            #elle prend la valeur 1
print(Resultat)           #affichage de la variable Resultat
```

1. Types et méthodes en informatique

a) Les types

Les données entrées et manipulées par Python se voient attribuer ce qu'on appelle **un type**. C'est ce qui spécifie à l'ordinateur l'interprétation de cette donnée : est-ce un entier, un réel, une liste d'objets..? Pour l'ordinateur, tout ça n'est qu'une suite de 0 et de 1, le type lui permet de l'interpréter comme on le veut.

Méthode globale : La fonction type renvoie le type de son argument.

Exercice 4. Taper valider observer commenter (dans l'invite de commande):

```
>>> type(10)
>>> type(10.0)
>>> type("Hello world")
>>> type(1==2)
>>> type(1,2)
>>> type((1,2))
>>> type([1,2])
>>> Type([1,2])
```

b) Les méthodes

Chaque type vient avec ce qu'on appelle ses **méthodes.** Ce sont des fonctions définies pour chaque type d'objet, qui permettent de faire des manipulations de base. Un programme informatique va s'appuyer sur ces méthodes pour effectuer les transformations voulues.

Exercice 5. Même consigne.

```
>>> 1+2
>>> 1.0 + 2.0
>>> L=[1,2]
>>> L.append(3)
>>> L
>>> "Hello" + "World"
>>> "Hello" + " " + "World"
```

Remarque. Pour la liste des méthodes globales en Python, tapez Python Built in Functions sur Google et regardez le premier lien: https://docs.python.org/3/library/functions.html.

Remarque. Pour vraiment apprendre à coder, vous devez apprendre à trouver les réponses à vos questions (sur un moteur de recherche par exemple). En général, la documentation Python dispose de tout ce dont on a besoin. Voir https://docs.python.org/3/library/index.html

Gardez toujours un onglet ouvert avec cette page en TP.

2. Liste des types avec leurs méthodes en `Python`

Cette exposition est incomplète, mais vous donnera assez d'éléments pour résoudre vos exercices ou, le cas échéant, comprendre la documentation `Python`.

a) Integers, abrev : int

Ce sont les nombres entiers. Les méthodes de bases incluent les opérations +,-,*. On a aussi l'opération de mise à la puissance **. Les entiers viennent aussi avec la division euclidienne.

Théorème 1. (de la division euclidienne) *Soient a et b deux entiers, avec $b \neq 0$. Alors, il existe d'uniques entiers q et r tels que*

$$a = bq + r$$

et $0 \leq r \leq |b| - 1$. q et r sont appelés respectivement le quotient et le reste de la division euclidienne de a par b.

L'opération `a//b` donne le quotient de la division euclidienne de a par b. L'opération `a%b` donne son reste.

La méthode `float` transforme un entier (comme 2) en le nombre réel lui correspondant (ici, ça serait 2.0).

b) floating-point numbers, abrev : float

Les nombres flottants sont une incarnation informatique des réels. Et je dis bien **une**, car modéliser les réels en informatique est un vrai problème. En effet, comment faire mémoriser toutes les décimales de π à un ordinateur n'ayant qu'une mémoire finie? Les nombres flottants représentent les réels à une précision fixée (et paramétrable). Donc, ce sont concrètement des nombres décimaux avec un nombre (paramétrable) fixé de décimales après la virgule.
Attention : En `Python`, la virgule numérique est signifiée par le point.

Remarque. Tapez 1/3 dans l'invite de commande, que remarquez vous?
Les opérations +,-,*,** des entiers sont aussi disponibles pour les flottants. L'opération de division / représente la division. Si vous effectuez une division entre deux `int` avec /, `Python` les transforme en `float` et fait la division en tant que `float`. On dispose aussi des opérateurs <, <=, >, >= de comparaison.

La méthode `float` converti son argument en flottant dès que ça fait sens.

Exercice 6. Même consigne.

```
>>> 2**3+4.5
>>> type("2.0")
>>> type(2.0)
>>> float("2.0")
>>> type(float("2.0"))
>>> 3<2
>>> 2<3
```

c) Strings, abrev : str

Les chaines de caractères est le type de donné correspondant au texte. Il est délimité par les guillemets (" ou '). Ainsi, `"2"` représente *le texte 2*, `"3.34"` représente *le texte 3.34*, etc.

Les méthodes sur ce type sont nombreuses. La plus utile : + est un opérateur de concaténation.

Exercice 7. Même consigne.

```
>>> "a"+"b"
>>> x='Hello World'
>>> x="variable called 'x' was 'Hello World'"
>>> print(x)
>>> 2*x
>>> x.capitalize()
>>> x
>>> y=x.capitalize()
```

```
>>> y
```

Remarque. Il y a un jeu de priorité entre les guillemets. Dans les guillemets ", on peut utiliser les guillemets ' comme texte.

Remarque. Beaucoup de méthodes se présentent comme une fonction à mettre en suffixe de leur argument. Certaines modifient l'argument, d'autres non.

d) Booleans, abrev : bool

Les booléens, ce sont les valeurs de vérité. Il y en a deux : `True` et `False`. Ce type de donnée permet de faire des opérations logiques et est donc omniprésent.

```
>>> 2>3
>>> type(2>3)
>>> 1==2
>>> 1=2
>>> True and True
>>> 2>3 or 3<2
>>> not 2>3
>>> (not 2>3) and (1==2 or 4==2*2)
```

e) Tuple and Lists

Le type **tuple** est un type servant juste a considérer des listes de données, mais avec très peu de méthodes. Un tuple est délimité par des parenthèses, et les données sont séparées par une virgule. Ils sont dits immuables : on ne peut réaffecter la valeur d'une entrée.

Au contraire, les **listes** sont très utilisées pour faire des opérations sur une "liste de données". Une liste est délimitée par des crochets `[, ]` et les données encore séparées par des virgules. On les manipule très facilement.
Remarque : Lists and tuple entries are indexed by integers, *starting at 0.*
Même exercice :

```
>>> x=(2,3)
>>> type(x)
>>> y=x+(4,5,'ecg')
>>> y
>>> y[1]
>>> y[1]=8
```

Pour les listes maintenant :

```
>>> x=[2,3]
>>> type(x)
>>> y=x+[4,5]
>>> y
>>> y[1]
>>> y[1]=8
>>> y
>>> y.append('1a')
>>> y
>>> len(y)
```

3. Variables en informatique

En informatique, pour manipuler des données, on les enregistre dans des **variables** (une case mémoire de l'ordinateur) puis on se réfère à ces variables à l'aide d'un nom qu'on a choisi. **Le symbole = sert, en informatique, à attribuer une variable. Il n'est pas symétrique et s'utilise exclusivement de la façon suivante** :

```
NomDeLavariable = "Valeur de la variable"
```

On peut utiliser le type `tuple` pour faire plusieurs affectations d'un coups. Attention à ne pas confondre nom de variable et valeurs. Essayez ça:

```
>>> x=2
```

```
>>> x=x+1
>>> print(x)
>>> print("x")
>>> print(2*x+1)
>>> print(2*"x"+1)
>>> a,b=3,"hi"
>>> print(b," there, variable 'a' also has a value :", a)
```

Remarque. La fonction `print()` demande à `Python` **d'afficher** ses arguments, rien de plus. Quels sont les 3 arguments dans l'utilisation de la fonction `print` ci-dessus? Cette fonction peut prendre plusieurs arguments. Quels sont les 3 arguments dans l'utilisation de la fonction `print` ci-dessus?

Exercice 8. Créez une variable `x` de type `str` contenant votre prénom répété 13 fois.

La fonction `input()` affiche le message donné, en entrée, attend que l'utilisateur entre quelque chose au clavier, validé par Entrée, et **renvoie** cette donné sous forme de texte.

Exercice 9. Dans l'**éditeur de code**, tapez ces lignes, puis lancer votre programme. Le programme ne marche pas, pourquoi ? Le corriger à l'aide de la fonction `float()`, voir exercice 6.

```
a=input("Entrez l'année actuelle")
n=input("Entrez votre année de naissance")
age = a - n
print("A quelque mois près, vous avez : ", age, "ans.")
```

4. Tests et boucles en informatique

Les tests et les boucles sont un élément indispensable de la programmation. Ils permettent d'exécuter un bout de votre code d'une certaine façon. Pour les tests, c'est si une certaine condition est vérifiée (type `bool`). Pour les boucles, elles permettent de répéter l'exécution d'un bout de code. Un élément très caractéristique de `Python` est que **l'indentation joue le rôle de syntaxe.** La syntaxe, c'est les règles d'écriture qui permettent à `Python` de savoir ce qu'il est en train de lire. L'indentation, c'est l'utilisation de la tabulation.

a) if : elif : else:

Prenons le code suivant (à copier et exécuter).

```
x=input("Aimez vous l'informatique? Répondez O/N")   #Attend une réponse de l'utilisateur.
if x=="O":                                           #Si l'utilisateur a répondu oui :
        print("Très bien.")
else:                                                #Sinon :
        print("Savez-vous ce qu'on dit de ceux qui ne changent pas d'avis?..")
```

Pour utiliser le test `if`, on respecte la syntaxe suivante, ou :

```
if CONDITION1 :
        ligne à exécuter conditionnellement 1
        ligne à exécuter conditionnellement 2
        ...
elif CONDITION2 :
        ligne a exécuter conditionnellement 1
        ...
elif CONDITION 3:
        ...
else :
        ...
```

où :

- CONDITION1, CONDITION2,... sont des expressions devant renvoyer un booléen (True/False),
- on n'oublie pas le deux-points `:` après les conditions,
- l'espace avant les lignes à exécuter conditionnellement est d'une tabulation exactement, soit 4 espaces. Les espaces en début de ligne comptent beaucoup pour `Python`.

- `elif` means “else if” ("sinon, si"). Ce teste sera effectué seulement si les *if* et *elif* précédents ont échoués (c'est-à-dire si la condition correspondante était `False`).
- `else:` ("sinon") donne le code qui sera lu "dans tous les autres cas",
- les `...` ne font pas partie de la syntaxe, je les mets pour signifier qu'on peut continuer si longtemps qu'on veut.

Remarque. On peut très bien effectuer un test conditionnel avec uniquement un `if`, sans `elif` et `else`.

b) Les boucles for

Pour répéter un bout de code, on utilise une boucle for.

```
x=0
for i in range(10):
        x=x+i
print("La somme des entiers de 0 à 9 est", x)
```

La syntaxe est la suivante, où :

- `NomDeVariable` est un nom de variable que l'on peut choisir. La variable ne doit pas être attribuée.
- `Iterateur` est ce qui permet d'indiquer à la machine la manière dont le code sera répété. On n'oublie pas le deux-points.
- La même règle d'indentation que pour le `if` s'applique.

```
for NomDeVariable in Iterateur :
        Ligne à répéter 1
        ...
```

À chaque répétition, une variable locale NomDeVariable est crée et prends la valeur correspondante donnée par l'itérateur. Vous pouvez bien sûr vous en servir. On verra dans l'année d'autres structures de boucle. Exécutez ces codes un par un (séparés par les dièses) et essayez de comprendre. Commentez ici.

```
for t in range(100):
        print("bonjour")
###
for i in range(10):
        print("Voici une itération du code. La variable i contient la valeur suivante : ", i)
###
x=1
for i in range(2,10):
        print("Valeur de i ", i, " - valeur de x : ", x)
        x=x+1
###
x=1
for i in range(2,3,15):
        print("Valeur de i ", i, " - valeur de x : ", x)
        x=x+1
###
x=1
for i in range(0,2,10):
        print("Valeur de i ", i, " - valeur de x : ", x)
        x=x+i
###
for i in [2,4,5]:
        print(i)
###
for i in "HelloWorld":
        print(i)
###
```

5. Les commentaires

Il est important de commenter son code pour expliquer le rôle des lignes importantes. Le symbole de dièse dit à Python de ne pas lire la fin de la ligne, et permet donc de mettre vos remarques dans votre fichier `.py`. Voir

l'exemple du 4.a).

III. Premiers exercices

Exercice 10. Qu'affiche le programme suivant? Prédire le résultat sans copier le code.

```
x=10
y=x^2
h=x+y+x/10
if y<x:
        print(y+x)
else:
        print(h)
```

Exercice 11. Écrire un programme permettant de calculer la somme $1+3+5+7+\ldots+999$.

Exercice 12. Soit (u_n) la suite donnée par la relation de récurrence suivante : $u_0 = 1$ et pour tout entier n, $u_{n+1} = u_n^2 - \dfrac{1}{2}$. Calculer (une valeur approchée de) u_{100}.

Exercice 13. Écrire un programme permettant, à partir d'un entier positif n donné en entrée, de calculer successivement :

- la partie entière $\lfloor \frac{n}{2} \rfloor$ de $\dfrac{n}{2}$, notée d dans la suite. Indice : `int(3.5)` renvoie l'entier `3` : on peut ici utiliser la fonction `int()` pour avoir une partie entière.
- l'entier $d! = 1 \times 2 \times 3 \times ... \times d$,
- l'entier $(n-d)! = 1 \times 2 \times 3 \times ... \times (n-d)$
- l'entier $n! = 1 \times 2 \times 3 \times ... \times n$,
- l'entier $\binom{n}{d} = \dfrac{n!}{d!(n-d)!}$

puis d'afficher cet entier $\binom{n}{d}$.