TP de Python numéro 1 - Prise en main

*Semaine du **12 septembre 2024**.*

Un premier TP de prise en main de `Python` avec l'IDE Pyzo.

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.

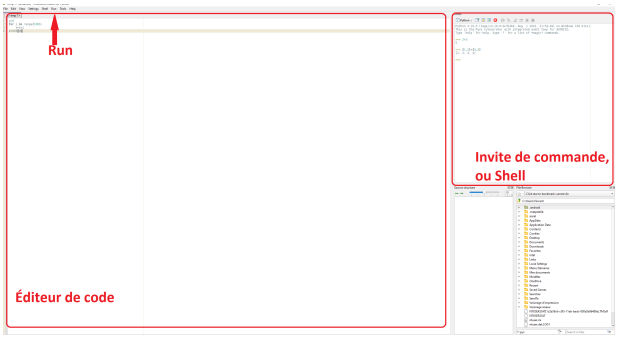

Figure 1: *Vous pouvez déjà identifier deux espaces : l'invite de commande et l'éditeur de code.*

a) L'invite de commande

Dans **l'invite de commande**, toute ligne que vous tapez validée par `Entrée` est immédiatement interprétée par `Python`, à l'instar d'une calculatrice. 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 une par une. Puis commentez sur cette feuille :

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

b) L'éditeur de code

Dans **l'éditeur de code**, vous pouvez écrire 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. Le texte tapé après le symbole `#` n'est pas lu par la machine. Cela permet de commenter notre code, c'est à dire d'ajouter du texte expliquant à un humain le fonctionnement du 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 et aux tabulations : il prends en compte la différence entre majuscule et minuscule, ainsi que le nombre d'espaces en début de ligne.
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 :

- Entrée(s) : 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
```

Exercice 4.
1. Copier le code précédent dans l'éditeur de code.
2. Exécuter ce code (Run > Run file as script ou execute file)
3. Changez le contenu de la variable `n` donnée en première ligne, et exécutez le code à nouveau.

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 5. 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 ou opérations 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 à priori plus complexes.

Exercice 6. Même consigne.

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

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`

Vous devriez toujours garder 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 7. (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*

$$\begin{cases} a = bq + r \\ 0 \leq r \leq |b| - 1 \end{cases}.$$

Les entiers 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.

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 8. Même consigne.

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

c) String, abrev : str

Le type `String` ("chaîne de caractère") est le type de donnée 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 ("mise bout à bout") et * permet de faire des répétitions.

Exercice 9. 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

Les **listes** permettent de constituer des listes de données.

C'est un type très pratique : une liste peut être modifiée de nombreuses manières (on peut changer la valeur d'une donnée, rallonger ou raccourcir une liste...). Une liste est délimitée par des crochets `[, ]` et les données de la liste sont séparées par une virgule.

Même exercice que précédemment :

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

Les `Tuples` ont un rôle similaire, mais sont "plus rigides" : on ne peut pas modifier un tuple autant qu'une liste ne le permet. Cette rigidité est utilisée dans certains contextes.

Un tuple est limité par des parenthèses (qui sont parfois facultatives), et les données sont encore séparées par une virgule.

Même exercice :

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

Remarque : Lists and tuple entries are indexed by integers, *starting at 0.*
Cela signifie que la première donnée d'une liste `L` est désignée par `L[0]`, et non pas par `L[1]`. Idem pour les tuples.

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.

Absolument fondamental : le symbole =

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`

Exemples :

```
>>> x=2
>>> x=x+1
>>> print(x)
>>> print("x")
>>> print(2*x+1)
```

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

```
>>> a,b=3,"hi"
>>> a
>>> b
>>> print(b," there, variable 'a' also has a value :", a)
```

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

Exercice 10. 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 11. 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 8.

```
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 des éléments indispensables de la programmation. Ils permettent d'exécuter un "bout de code" (appelé **bloc** à partir de maintenant) d'une certaine façon.

Un test se déclare avec le mot clé `if` et permet de demander à `Python` de lire un bloc donné uniquement si une condition spécifiée (à l'aide du type `bool`) est vraie (`True`).

Une boucle se déclare avec l'un des mot clé `for` ou `while` et demande à `Python` de répéter la lecture selon des conditions spécifiées.

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 (4 espaces) en début de ligne.

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("Changez vite d'avis !")
```

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

```
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 données,
- 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` ni `else`.

Exercice 12. 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:
        h=h-10
        print(y+x)
else:
        h=2*h
        print(h)
```

Exercice 13. À l'aide d'un test `if`, écrire un programme permettant de savoir si un nombre donné x est strictement supérieur à $x^2 - x + 1$, en affichant "Oui" si c'est le cas, et "Non" sinon.

b) Les boucles for

Pour répéter un bloc, 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 :

```
for NomDeVariable in Iterateur :
        Ligne à répéter 1
        Ligne à répéter 2
        ...
```

où :

- `NomDeVariable` est un nom de variable que l'on peut choisir parmi les noms disponibles.
- `Iterateur` est ce qui permet d'indiquer à `Python` la manière dont le bloc sera répété. On n'oublie pas le deux-points.
- La même règle d'indentation que pour le `if` s'applique.

Voici les itérateurs les plus classiques :

- Si `d` ("début") et `f` ("fin") sont des entiers, l'itérateur `range(d,f)` lira le bloc donné pour chaque valeur de la variable `NomDeVariable` entre `d` et `f-1` (**attention au -1**).
- Si `f` est un entier, l'itérateur `range(f)` a le même effet que `range(0,f)`.
- Si `p` est un entier ("pas"), l'itérateur `range(d,f,p)` lira le bloc donné pour chaque valeur de la variable `NomDeVariable` entre `d` et `f-1`, mais en prenant faisant eds "sauts" de `p` entre ces valeurs.
- Une liste peut être utilisée comme itérateur (très courant). Dans ce cas, le bloc est répété pour chaque élément de la liste (dans l'ordre), et la variable `NomDeVariable` prends comme valeur ces éléments, un à un.
- Plus curieux : une chaîne de caractère peut être utilisée, comme si c'était la liste de ses lettres.

À chaque répétition, une variable NomDeVariable est crée et prends une valeur donnée par l'itérateur.

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+4
###
x=1
for i in range(2,15,3):
        print("Valeur de i ", i, " - valeur de x : ", x)
        x=x+4
###
x=1
for i in range(1,15,10):
        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 14. 1. Recopier et compléter le code suivant pour qu'il calcule et affiche la moyenne des notes données dans la liste en première ligne.

```
x=[11,14,8,13,10.5,17,9,15]
m=0
for n in ... :
        m=...
l=... #nombre de notes
m=m/l
print(m)
```

2. Compléter ce code pour qu'il affiche "Mention" si cette moyenne est supérieure à 12, "Incertain" si cette moyenne est entre 8 et 12, et "Refus" sinon.
3. Compléter le code précédent pour qu'il affiche en plus la dernière note de la liste `x` dans le cas où la moyenne est entre 8 et 12.

   Remarque : Si `L` désigne une liste, la commande `L[-1]` renvoie le dernier élément de `L`.

Exercice 15. 1. Écrire un programme permettant de calculer la somme $1+3+5+7+\ldots+999$.

2. Écrire un programme permettant de calculer la somme $10+12+14+16+\ldots+888$.
3. Écrire un programme permettant de calculer la somme $1^2+2^2+\ldots+56^2$.
4. Écrire un programme permettant de calculer le produit $(1+2)(1+2+3)(1+2+3+4)\ldots(1+2+3+\ldots+42)$.

Exercice 16. 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 u_{100} à l'aide d'une boucle.

Exercice 17. 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 $\frac{n}{2}$, 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 $\binom{n}{d}$.

Exercice 18. Soit $(u_n)_{n\in\mathbb{N}}$ la suite réelle donnée par $u_0 = 0$ et pour tout entier n : $u_{n+1} = 3u_n + 2^n$.

1. À l'aide d'une boucle `for` , calculer u_{10}, u_{45} et u_{100}.
2. Calculer $3^{10} - 2^{10}$, puis $3^{45} - 2^{45}$ et $3^{100} - 2^{100}$. Conjecturer une relation vis à vis de la suite précédente et la démontrer brièvement sur papier.

Sommes et produits

On rappelle les notations suivantes, pour toute suite réelle $(u_n)_{n\in\mathbb{N}}$. Pour tout entier n :

$$\sum_{k=0}^{n} u_k = u_0 + u_1 + u_2 + \ldots + u_n$$

$$\prod_{k=0}^{n} u_k = u_0 u_1 u_2 \ldots u_n.$$

En particulier, on utilisera $\displaystyle\sum_{k=0}^{n+1} u_k = \left(\sum_{k=0}^{n} u_k\right) + u_{n+1}$ et la formule analogue pour le produit.

Exercice 19. Sans utiliser l'ordinateur, quelle est la somme affichée par le code ci-dessous ?

```
S=0  #Avant la boucle, S=0
for i in range(0,10):
        S=S+i
print(S)
```

Exercice 20. • Calculer la valeur des sommes et produits suivants (pour $n = 10, 50, 100$ si la formule dépend de n). Gardez une trace de votre travail en travaillant dans l'éditeur de code.

1. $\displaystyle\sum_{i=0}^{n} i^2$
2. $\displaystyle\sum_{i=4}^{57} \left(-\frac{1}{2}\right)^i$
3. $\displaystyle\prod_{k=0}^{n} k$
4. $\displaystyle\prod_{k=1}^{n} \frac{i}{i+1}$

- Corriger l'énoncé pour rendre la question 3 intéressante et adapter votre code.
- Conjecturer et essayer de démontrer brièvement une égalité liée au dernier produit 4. Pour la conjecture, calculer ce produit pour $n = 2, 5, 10$.

Exercice 21. On s'intéresse aux restes dans la division euclidienne par 7 des nombres de la forme $8^n + 7$, pour n entier. On va utiliser `Python` pour faire des tests et conjecturer un résultat.

1. Faire afficher ces restes pour n variant de 0 à 10. Émettre une conjecture.
2. Vérifier votre conjecture un peu plus sérieusement, en codant un programme qui teste votre conjecture pour n allant jusqu'à 1000, et affiche un message adéquat s'il trouve un contre exemple (utiliser `if`).

Exercice (à la maison) : démontrer votre conjecture par récurrence.