# Ch 1. Prise en main de Python.

*Ce chapitre est une activité à réaliser avant le TP01 d'informatique en classe.*

## 1 Découverte de quelques types dans l'interpréteur

Lancer PYZO.
Une fenêtre s'ouvre sur votre écran partagée en 3 zones, il est nécessaire d'ouvrir le shell en choisissant la version de python, alors quelques lignes de message introductif s'inscrivent. C'est l'interpréteur de Python, qu'on appelle aussi le shell.
Les symboles `>>>` invitent à taper quelque chose (invite de commande).

**Fonctionnement** : tapez une *instruction*, par exemple $1+1$, et tapez "Entrée" :

*Python shell*
```
>>> 1 + 1
2
```

### 1.a Différents types de nombres

Le shell de Python peut s'utiliser comme une calculatrice pour manipuler des nombres. Selon la nature (entiers, réels, complexes) des nombres que l'on souhaite manipuler, il y aura différents *types* sous python pour les représenter.

Ci-dessus, on a manipulé `1` qui est de type `integer` (*entier* en anglais).
On manipule aussi les "nombres à virgule", dits nombres *flottants*, dont le type s'appelle `float` en python :

*Python shell*
```
>>> 3.1 + 10.09
13.19
>>> type(2.7)
<type 'float'>
```

⚠ Bien remarquer que l'on utilise la convention anglosaxonne : un point et non une virgule.

**Exercice 1.**

- Taper les deux instructions suivantes : `3` , `3.0` puis `'3'`
  puis demander le type de ces données, commenter :

- Taper les trois instructions suivantes : `3.` `0.1` `.1`
  Commenter :

*DONNEES*

Ce qu'il faut retenir :

**Exercice 2.** Testez les opérateurs suivants avec des entiers et/ou des flottants :

`-  *  /  //  %  ^  **`

En particulier, que dire du résultat et du type du résultat pour les opérateurs `/`, `//` et `%`, par exemple comment relier les résultats obtenus de ces deux opérations : `9//2` et `9%2` ? L'opérateur `^` renvoie-t-il le résultat attendu ? Que ce passe-t-il en cas d'opération entre un entier et un flottant ?

*DONNEES*

Ce qu'il faut retenir :
+;-;*;/:
//;% :
^ ;** :

Les fonctions `float` et `int` permettent d'obtenir une donnée du type voulu. Taper les instructions suivantes, où l'on voit des premiers exemples d'*affectations*, ici des *variables* `x` et `y` :

*Python shell*
```
>>> x = 42
>>> x
.................. à compléter !
>>> float(x)
..................
>>> y = -3.67
>>> y
..................
>>> int(y)
..................
```

⚠ `int` appliqué à un flottant ne fait pas un arrondi, ni ne calcule la partie entière : cela donne juste l'entier qui apparaît dans l'écriture décimale du flottant.

## 1.b Les booléens

Une variable de type booléen ne peut prendre que deux valeurs : `True` ou `False`. Il s'agit souvent de propositions logiques plus ou moins longues que python peut évaluer. Quelques exemples :

*Python shell*
```
>>> 1 > 0
True
>>> a = (1 > 3)
>>> a
False
>>> type(a)
<type 'bool'>
```

```
>>> x = 3.14
>>> y = 3.1
>>> x == y
False
```

Les opérateurs de comparaison disponibles sont :

DONNEES

| | |
|---|---|
| `x == y` | $x$ est égal à $y$ |
| `x != y` | $x$ est différent de $y$ |
| `x > y` | $x$ est strictement supérieur à $y$ |
| `x < y` | $x$ est strictement inférieur à $y$ |
| `x >= y` | $x$ est supérieur ou égal à $y$ |
| `x <= y` | $x$ est inférieur ou égal à $y$ |

Bien noter la différence entre `=` (pour affecter une variable se trouvant à gauche du signe) et `==` (pour définir une égalité qui sera un booléen).
Pour combiner des variables booléennes, on dispose de trois opérateurs logiques : `not`, `and`, `or`.

**Exercice 3.** Créer une variable `x` égale à l'entier 3, et tester les trois opérateurs logiques ci-dessus sur les expressions `x < 4.1` et `x % 2 == 0`.

Mathématiquement, à quoi correspond le booléen `x % 2 == 0` ?

On rappelle qu'une proposition $P$ *ou* $Q$ est vraie si et seulement si l'une au moins des propositions est vraie, éventuellement les deux. Ce n'est pas le "ou exclusif" que l'on trouve par exemple sur les menus de restaurants : "Fromage ou dessert" (on ne peut pas avoir à la fois le fromage et le dessert... avec le "ou logique", c'est tout-à-fait admis.)

## 1.c Les chaînes de caractères

Python permet aussi de manipuler du texte. Le type utilisé s'appelle une chaîne de caractère : *string* en anglais, abrégé en `str` en Python. Il suffit de délimiter le texte par des apostrophes (*quote*) ou par des guillemets (*double quote*) :

*Python shell*
```
>>> c = 'ptsi'
>>> d = "bonjour à tous"
```

On peut avoir accès aux caractères qui composent une chaîne de caractères ; attention, en Python, ils sont numérotés à partir de 0, c'est-à-dire que le premier caractère est d'indice 0.

Noter les résultats et commenter :

*Python shell*
```
>>> c[0]
..................
>>> c[2]
..................
>>> c[-1]
..................
>>> c[0] = 'm'
..................
```

DONNEES

Trois autres choses à retenir sur le type `string` :

- La longueur d'une chaîne de caractères, c'est-à-dire le nombre de caractères qui la composent, s'obtient avec la fonction `len`.
  Tester avec `c` et `d`.
- On peut concaténer des chaînes de caractères (c'est-à-dire les "accoler") à l'aide de `+`.
- Comme pour les fonctions `int` et `float`, il existe une fonction `str` qui permet d'obtenir un objet de type string.

**Exercice 4.**

- Créer une chaîne de caractère `e` obtenue en concaténant `c` et `d`.
- Tester les instructions suivantes ; comment modifier la dernière pour qu'il n'y ait pas de message d'erreur ?

*Python shell*
```
>>> numero = 2
>>> f = c + numero
```

*Remarque :* Le fait qu'un même opérateur (ici `+`) puisse fonctionner différemment en fonction du contexte illustre le mécanisme appelé *surcharge des opérateurs*.

*Remarque : Pour ne pas avoir à tout recopier à chaque ligne du shell en cas d'erreur...*
Sous PYZO (windows), Avec la flèche ↑ (éventuellement plusieurs fois successivement), on peut remonter dans l'historique de commandes.
Sous IDLE, Se placer sur la ligne à répéter et "entrée".

# 2 Prise en main en mode "exécution d'un fichier"

Fermer la fenêtre du shell en cliquant sur la croix blanche sur fond rouge. Relancer le shell à partir de l'onglet "no shell selected". Tout ce qu'on a écrit est perdu...
Pour écrire de véritables programmes et les conserver, il faut créer un fichier python exécutable, ce qu'on appelle aussi un "script".
Pour cela, à partir de l'interpréteur :

aller dans "File" et cliquer sur "New"

(plus rapide : CTRL+N).

Avant toute chose, enregistrer ce fichier dans votre répertoire "TP0" sous le nom TP0.py ; il ne faut pas oublier de préciser l'extension .py ("File" puis "Save" ou CTRL+S).

Taper dans le fichier les lignes suivantes :

```
2 * 3
print(3 * 3)
a = 4 * 3
print(a + 2)
```

Sauvegarder, puis exécuter (*run* en anglais) :

soit en tapant CTRL+E, soit en cliquant sur "Execute file" dans le menu "Run"

Si votre script fait appel à des fichiers stockés dans le même dossier que votre script, la première exécution doit repérer le chemin d'accès aux données, dans ce cas là, exécuter :

en cliquant droit sur l'onglet de votre script et en sélectionnant "Run file"

Commenter :

**Exercice 5.** Créez dans votre script deux variables `x` et `y` contenant des entiers de votre choix.

**1°)** Complétez votre script pour qu'à l'exécution, il s'affiche dans le shell le booléen disant si `x` est pair ou non.

**2°)** Complétez votre script pour qu'à l'exécution, il s'affiche dans le shell le booléen disant si `x` et `y` sont de même parité ou non.

**Exercice 6.** Créez dans votre script deux variables `prenom` et `nom` de type chaîne de caractère. Complétez votre script pour qu'à l'exécution, il s'affiche vos initiales (par exemple `LM` pour Léa Martin).

Après ce premier test et ces deux exercices, votre script ressemble à peu près à cela :

```
2 * 3
print(3 * 3)
a = 4 * 3
print(a + 2)
x = 2
y = 5
...ce que vous avez répondu pour l'exo 5...
prenom = 'Léa'
nom = 'Martin'
...ce que vous avez répondu pour l'exo 6...
```

Ainsi, on ne voit pas bien la délimitation des différents exercices ; de plus, lorsqu'on exécute ce script, on a les affichages "parasites" du test et de tous les exercices, alors qu'on aimerait passer à autre chose...Il ne faut pas effacer votre travail pour autant !
Il convient donc d'utiliser des commentaires à l'aide du symbole # ; tout ce qui suit sur la même ligne ne sera pas "lu". On s'en sert pour améliorer la présentation, et pour supprimer les affichages intempestifs lors de l'exécution. Par exemple, il convient de transformer notre fichier ainsi :

```
############ TP 2 Prise en main de Python ############

##### Premier test
```

```
2 * 5
#print(3 * 5)
a = 4 * 5
#print(a + 2)

##### Exercice 5
x = 2
y = 5
...ce que vous avez répondu pour l'exo 5, que vous pouvez mettre en commentaire !

##### Exercice 6
prenom = 'Léa'
nom = 'Martin'
...ce que vous avez répondu pour l'exo 6, que vous pouvez mettre en commentaire !
```

On peut mettre tout un paragraphe en commentaire en le sélectionnant, puis en allant dans le menu "Format" et en cliquant sur "Comment Out Region". L'opération inverse se fait avec "Uncomment Region".

Lorsqu'on ferme PYZO et qu'on le relance, votre fichier TP02.py se retrouvera facilement en cliquant sur "Open" dans le menu "File", ou bien en faisant CTRL+O.

**Exercice 7.** Toujours en se servant des variables `prenom` et `nom`, écrire les instructions permettant d'afficher par exemple le message : `'Bonjour, Léa Martin !'` lorsque `prenom` contient `'Léa'` et `nom` contient `'Martin'`.

# 3 Les fonctions

Nous avons déjà rencontré des fonctions python prédéfinies : `type`, `int`, `len`, `print`...

## 3.a Utilisation de l'aide

Lorsqu'on écrit le nom d'une fonction `fct` dans l'interpréteur et sa parenthèse ouvrante, des informations s'affichent (les arguments attendus et optionnels, parfois le type du résultat...).
Pour obtenir plus de détails, on peut aussi taper `help(fct)`.

**Exercice 8.** À l'aide de ces deux techniques :

**1°)** Vérifier que la fonction `abs` renvoie bien la valeur absolue.

**2°)** Comprendre le fonctionnement de la fonction `round`, et tester avec des exemples pour vérifier.

On peut aussi consulter l'aide sur internet notamment sur `docs.python.org`, mais il faut savoir trier les informations.

## 3.b Fonctions issues d'une bibliothèque

Souvent, nous aurons besoin de fonctions non disponibles par défaut, qu'il faut alors importer à partir d'une bibliothèque [1]. C'est le cas pour de nombreuses fonctions mathématiques ; par exemple :

```
Python shell
>>> cos(1)
Traceback (most recent call last):
   File "<pyshell#27>", line 1, in <module>
      cos(1)
NameError: name 'cos' is not defined
```

Importons la bibliothèque `math` ; voici une méthode possible :

1. Remarque : en anglais, bibliothèque se dit *library*.

*Python shell*

```
>>> import math as m    # ceci signifie "importer la bibliothèque math sous le nom m"
>>> m.cos(1)
0.5403023058681398
```

Quelques fonctions mathématiques usuelles sont dans cette bibliothèque :
`cos, sin, tan, exp, log` (logarithme népérien noté ln en mathématiques),
`sqrt` (racine carrée : "**sq**uare **r**oo**t**"),
`floor` (partie entière habituelle : `floor(x)` est l'entier inférieur le plus proche de `x`),
`ceil` (partie entière supérieure : `ceil(x)` est l'entier supérieur le plus proche de `x`).

## 3.c Définir vos propres fonctions

Vous pouvez définir vos propres fonctions. Pour cela il faut se placer dans le script TP0.py et plus dans le shell. Une fonction se définit de la façon suivante :

```
def f(x): # Def appelle une fonction et f indique son nom
     return(x**2)
```

*DONNEES*

On dit que la variable `x` est l'*argument* de la fonction `f`, et que cette fonction *renvoie* (ou *retourne*) $x^2$.

Après avoir écrit les `:`, lors du passage à la ligne, il y a une indentation automatique : elle est indispensable, elle permet à Python de savoir si ce qui fait partie de la définition de la fonction et ce qui n'en fait plus partie. C'était la même chose pour la structure `for`.

Après avoir défini une fonction, on peut l'évaluer en une valeur précise : par exemple, exécutez le script, et dans le shell, calculez $f(12)$.

Une fonction n'associe pas forcément un réel à un réel ; par exemple, la fonction suivante renvoie `True` ou `False` selon que l'argument entier `n` est pair ou non :

```
def g(n):
     return( (n % 2) == 0 )
```

Une fonction peut avoir plusieurs arguments, que l'on sépare alors par des virgules ; le bloc d'instruction peut être plus long :

```
def h(x,y):
     z = 2 * y
     return(x + z)
```

(On aurait cependant pu se contenter d'une seule instruction : `return(x + 2 * y)` )

*DONNEES*

La syntaxe générale d'une fonction est la suivante :

```
def nom_de_la_fonction(arguments):
     bloc d'instructions
```

Le chapitre 3 au cours de la séance de TP01 propose des applications qui portent sur l'utilisation de fonctions dans l'algorythme.