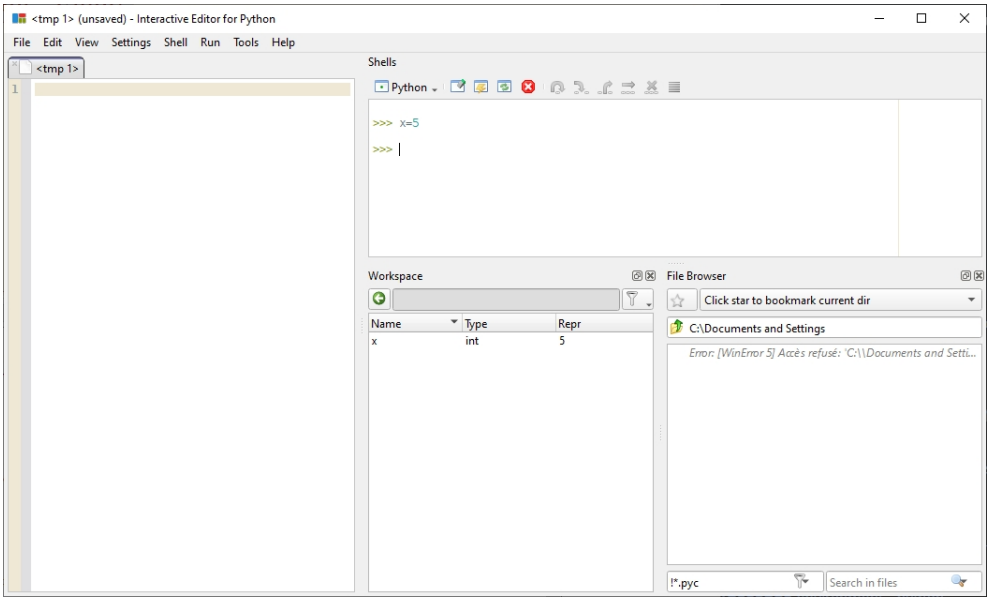
# INTRODUCTION À PYTHON

Pour les TP de cette année, nous utiliserons le logiciel **Pyzo**. Lancer Pyzo depuis le bureau. La fenêtre qui s'ouvre est divisée en plusieurs parties*.




* Il se peut qu'à la place du Workspace, vous voyez une autre fenêtre, telle que « source structure ». Pas de panique. On reviendra sur le Workspace plus bas.

## 1 Console et opérations simples

La *console* (ou *Shell*) est comme une calculatrice, on tape quelque chose, on valide avec Entrée et le résultat apparait. Essayer avec les instructions ci-dessous.

a) L'addition +
```
2+3
7+1.4
```

b) La soustraction -
```
5.5 - 1.1
-2-3
```
*(les espaces aident à relire)*

c) La multiplication *
```
2*3
1+0.5*2
(1+0.5)*2
```
*(* prioritaire sur + et -)*

d) La division /
```
3/4
1+2/3
1/(2*2)
1/2*2
```
*(idem pour /)*
*(pas de priorité entre * et /)*

e) La puissance (ou exponentiation) **
```
5**2
4**(1/2)
3*2**2
```
*(rappel : $\sqrt{x} = x^{1/2}$)*
*(** prioritaire sur * et /)*

f) La division entière (quotient de la div. euclidienne) //
```
10//3
8//2
```

g) Le reste de la division euclidenne %
```
10%3
8%2
```

h) La concaténation de listes et de *strings* (chaînes de caractères)
```
[1,5,7] + [2,1]
'Salut' + ' le monde'
```

## 2 Éditeur et variables

*L'éditeur* permet d'écrire des *scripts* : chaque ligne sera exécutée l'une après l'autre. Grâce à `print`, on peut afficher un résultat dans la console. Taper dans l'éditeur, puis compiler avec Ctrl + E.

```
print('Bonjour, humain')
```

```
a = 5
b = 12
c = a*b
print(c)
```

a, b et c sont des *variables* : ce sont des « boîtes » qui stockent une *valeur*. Attention à l'ordre : la syntaxe est toujours `<variable> = <valeur>`. Par exemple, `annee = 2022` mais par contre ~~`2022 = annee`~~.

**Question 1.** Est-ce qu'il y a une différence entre les instructions `a = b` et `b = a` ?

**Le Workspace.** En plus de la console et de l'éditeur, Pyzo permet de rajouter différentes « fenêtres-outils ». Pour afficher le Workspace, cliquer sur Tools » Workspace. Vous pouvez enlever des outils ou les repositionner comme avec une fenêtre normale. Le *Workspace* affiche toutes les variables, leur type (on y reviendra) et leur valeur associée. Cela peut être très pratique !

## 3 Si ... (alors) ... sinon ...

Si (`if`) une condition est vraie, alors on exécute une instruction, sinon (`else`) on en exécute une autre.

```
if condition :
    ...     # Ce bloc est exécuté si <condition> est vraie
    ...
else :
    ...     # Ce bloc est exécuté si <condition> est fausse
    ...
# fin du bloc

# tout ce qui se trouve après # est traité comme un commentaire, donc non compilé.
```

Le bloc `else` est optionnel.

**Exemple.** Recopier le script suivant et l'exécuter (avec Ctrl + E). Comparer pour différentes valeurs de a, b et c :

```
a = 11
b = 3
c = 17
if a + b == c :
    print("L'égalité est juste")    # On utilise "..." plutôt que '...' : pourquoi ?
else :
    print("L'égalité est fausse")
print(c)                            # Ceci s'affiche dans les deux cas
```

**Syntaxe d'un bloc `if`.** « : » marque le début du bloc. L'espace en début de ligne indique qu'on est dans le bloc. C'est ce qu'on appelle *l'indentation*. Pour indiquer qu'on sort du bloc, il faut enlever l'indentation en supprimant cet espace. Pyzo gère automatiquement l'indentation, mais s'il faut indenter manuellement, on utilisera la touche Tab →| ou bien 4 espaces ____.

**Différence entre = et == .** Le signe = sert à affecter une valeur à une variable. Le signe == teste une égalité et renvoie `True` (Vrai) ou `False` (Faux). Pour écrire la condition d'un bloc `if`, il ne faut donc pas utiliser = mais bien == !

**Question 2.** Est-ce qu'il y a une différence entre les instructions `a == b` et `b == a` ?

## 4 Boucle `for` et boucle `while`

Une boucle consiste à répéter un bloc d'instructions. La boucle `while` permet de répéter un bloc *tant que* une condition est vraie. Recopier et compiler les 3 scripts ci-dessous.

```
N = 3
while N > 0 :
    print(N)          # tant que N>0, le bloc s'exécute
    N = N-1
print('partez !')
```

La boucle `for` permet de répéter le bloc en changeant à chaque fois la valeur d'une variable.

```
for k in [2,0,2,2] :
    print('k vaut',k)
```

La fonction `range` permet de générer facilement une liste d'entiers.

```
for i in range(1,4) :    # /!\ range(1,n) correspond à [1,2,...,n-1]
    print('et',i)
print('zéro !')
```

**Astuce utile !** Si vous ne voulez pas compiler tout le script mais juste une partie, on peut le diviser en « cellules » (ou cells en anglais). Une cellule est délimitée par deux lignes de commentaires qui commencent par ##.

```
## Cellule pour "while"
N = 3
while N > 0 :    # Pour compiler cette cellule, placer le curseur dedans et Ctrl+Entrée
    ...
## Cellule pour "for" n°1
for k in [2,0,2,2] :
    ...
## Cellule pour "for" n°2
for i in range(1,4) :
    ...
```

**Question 3.** Quelles sont les valeurs finales de `N`, `k` et `i` dans les scripts ci-dessus ? *Essayer de répondre en regardant juste le script, avant de vérifier avec Pyzo.*

Bien souvent, on utilisera une variable pour la boucle, et une autre variable en dehors. Regarder le script ci-dessous.

```
s = 0
for i in range(5) :
    s = s + i
print(s)          # que calcule s ?
```

**Exercice 1.** Écrire un script qui affiche le produit des entiers de 5 à 15.

**Exercice 2.** Écrire un script qui affiche la somme des entiers *pairs* de 10 à 30.

# 5 Les fonctions

Les fonctions sont des scripts qui prennent un ou plusieurs *arguments* en entrée et retournent un résultat en fonction de ces arguments.

```
def somme(a,b,c) :
    return(a+b+c)
```

Exécuter le script ci-dessus, puis taper par exemple `somme(5,-8,7)` dans la console. Essayer aussi `somme([1,2],[3],[4,5,6])` et `somme("Vive ","la ","prépa")`. Commenter.

**Syntaxe d'une fonction.** Le mot-clé `def` signifie qu'on introduit une fonction. Ci-dessus la fonction s'appelle `somme` et prend 3 arguments. Le mot-clé `return` permet de choisir ce que renvoie la fonction. ATTENTION : lorsque la fonction est écrite, *ou chaque fois qu'elle est modifiée,* il faut exécuter le script pour « enregistrer » la fonction dans le Workspace.

On connait déjà plusieurs fonctions :

- La fonction `print(...)` permet d'afficher son argument dans la console.
- La fonction `range(m,n)`retourne la liste `[m,m+1,...,n-1]`.
- A noter que `range(n)` équivaut à `range(0,n)`. Ainsi, dans Python, une fonction peut donner des résultats différents selon le nombre d'arguments.

**Exercice 3.** Que retourne `range(m,n,p)` *? (Essayer avec des valeurs particulières).* Améliorer le script de l'exercice 2 en utilisant cette nouvelle fonction.

**Exemple.** Le script suivant retourne le nombre de diviseurs de `n`.

```
def diviseur(n) :
    c = 0
    for d in range(1,n+1) :
        if n % d == 0 :
            c = c + 1
    return c                # on peut omettre les parenthèses avec return
```

On notera qu'on peut « imbriquer » les blocs : à chaque nouvelle valeur de `d`, on teste la condition `n % d == 0`. L'instruction `return c` marque la fin des blocs `for` et `if`.

**Exercice 4.** Modifier la fonction ci-dessus pour que `diviseur(n)` retourne une liste de tous les diviseurs de `n`. *Cf plus bas pour la syntaxe des opérations élémentaires sur les listes.*

**Exercice 5.** Écrire une fonction `somme(L)` qui retourne la somme de tous les éléments de la liste `L`. *Cf plus bas pour la syntaxe des opérations élémentaires sur les listes.*

# 6 Exercices d'approfondissements (facultatif)

**Exercice 6.** Écrire une fonction `premier(n)` qui retourne la liste des `n` premiers nombres premiers. *On rappelle que 1 n'est pas un nombre premier.*

# 7 Syntaxe pour une liste

- Pour définir une liste : `L = [a,b,c]` (ou `a`, `b`, `c` sont trois variables), ou encore `L = []` pour une liste vide.
- Pour accéder à un élement de la liste : `L[0]` pour le premier élément, `L[1]` pour le second, etc.
- Pour ajouter un élément à une liste : `L.append(d)` ou encore `L = L + [d]`.
- Pour obtenir la taille de la liste : `len(L)`.