Les variables

On ne va pas aborder tous les types existants dans le langage Python, seuls les plus courants.

Par abus de langage ont dit qu'une variable « contient » une donnée.
En réalité, il serait plus précis de dire que **le nom de la variable fait référence à une donnée qui est stockée en mémoire** !

Chaque variable a un type selon la nature de la donnée à laquelle elle fait référence :

int → nombre entier

Ex : val = 4 → val fait référence à une donnée de type int : 4
On peut convertir un nombre au format texte (str) avec la fonction ***int(…)***
Exemple : valeur = int('123') → valeur « contiendra » le nombre 123

La fonction int(…) permet aussi de récupérer la **partie entière** d'un nombre décimal
Exemple : valeur = int (13.98) → valeur « contiendra » le nombre 13 !

Note : si vous souhaitez obtenir la **valeur entière la plus proche** d'un nombre décimal, il vaut mieux utiliser la fonction **round(…)**
Exemple : valeur = round(13.98) → valeur « contiendra » la valeur 14.

float → nombre décimal

val = 4. → Attention ici à la présence du point '.' après le 4 qui indique que val fait référence à une donnée de type float : 4. est équivalent à 4.0
*A noter : le **séparateur décimal** pour Python est **le point** et non la virgule !*

```
>>> 2/3 == 1 - 1/3
False
>>> 2/3
0.6666666666666666
>>> 1-1/3
0.6666666666666667
```

Note : Il est déconseillé de tester la valeur d'un float avec '==' pour une condition étant donné que sa valeur peut être approximative compte tenu de la limite de précision propre aux données numériques.

On peut utiliser les puissances de 10 avec l'utilisation de la lettre 'e' →

```
>>> print( 2e-3 )
  0.002
```

str → string ou chaîne de caractères

Les chaînes de caractères sont des listes de caractères immuables *(qui ne peuvent pas être modifiées).*

On peut convertir une donnée numérique en chaîne de caractères avec la fonction **str (…)** *(exemple ci-contre →)*

```
>>> type ( 3.14159 )
<class 'float'>
>>> phrase = str( 3.14159 )
>>> type( phrase )
<class 'str'>
>>> phrase
'3.14159'
```

La fonction **len (** *variable* **)** renvoie le nombre de caractères dans la chaîne de caractères stockée dans *variable* :

```
>>> len("La petite maison")
16
```

Pour **accéder à un caractère**, indépendamment des autres, comme avec une liste, on utilise son **[index]** dans la chaîne de caractères (Cf les listes).

```
>>> phrase = "La petite maison"
>>> type(phrase)
<class 'str'>
>>> phrase[0], phrase[1], phrase[10]
('L', 'a', 'm')
```

On peut découper une chaîne de caractères en liste de mots avec la fonction **.split (…)** si on utilise l'espace comme caractère de séparation.

```
>>> "ma ta ça notre votre leur".split(" ")
['ma', 'ta', 'ça', 'notre', 'votre', 'leur']
```

Les listes peuvent stocker tous les types de variables. Pour déclarer une variable de type list, on utilise **deux crochets […]**.
Dans une liste, deux données sont **séparées par une virgule**.

```
>>> liste = [ 'a', 'b', 'c', 'd' ]
>>> print(liste)
 ['a', 'b', 'c', 'd']
>>> liste[0]
'a'
>>> liste[2]
'c'
>>> liste [-1]
'd'
```

On peut **accéder à un élément** en particulier en utilisant son **index** *(sa position dans la liste)* L'index est un nombre entier placé entre deux crochets.
Attention, **les index commencent à 0** !
***A savoir**, un index négatif signifie que l'on commence par les derniers éléments de la liste.*

La fonction ***len (*** *liste* ***)*** *renvoie* ***le nombre d'éléments contenus dans la liste.***

```
>>> liste = [ 'a', 'b', 'c', 'd' ]
>>> longueur = len(liste)
>>> print( longueur )
 4
```

Quelques fonctions souvent utilisées avec les listes :

val = **len (** *ma_liste* **)**	Renvoie le nombre d'éléments contenus dans la liste.
ma_liste.**append (** *data* **)**	**Ajoute** la donnée *data* en fin de liste.
ma_liste.**insert (** index, data **)**	**Insère** la donnée *data* dans la liste à l'*index* précisé
val = ma_liste.**pop( )** *val = ma_liste*.**pop(** *index* **)**	Retire et renvoie la donnée placée en début de liste si aucun index n'est précisé. Si un index est précisé, c'est l'élément placé à l'index indiqué qui est retiré de la liste et renvoyé
val = ma_liste.**index(** *data* **)**	Si la liste contient *data*, cette fonction renvoie son index. *Seul le premier élément rencontré sera pris en compte.*
if *data* **in** *ma_liste* **:**	… **in** … **Tester si la donnée** *data* **est-elle dans la liste** Ce test renvoie True si la donnée est présente, sinon False.
Liste = **[** 1, 2, 3 **] + [** 'A', 'B' **]**	Liste contient ensuite **[** 1, 2, 3, 'A', 'B' **]** → concaténation
Liste = **[** 1, 2 **] *** 3	Liste contient ensuite **[** 1, 2, 1, 2, 1, 2 **]**
Pour **obtenir dans la console la liste des fonctions disponibles** utilisez la commande : **dir (** *ma_variable* **)**, *si celle-ci est de type list bien entendu* ! Ensuite, si nécessaire, la commande **help (** *ma_variable*.la_fonction() **)** affiche l'aide pour l'utiliser.	

Attention**, beaucoup de fonctions dites « builtin » peuvent faciliter le codage, exemples :* ***max(*ma_liste* ***)*** *→ renvoie la valeur la plus élevée ;* *ma_liste*.***remove(*** *data* ***)*** *→ retire la première donnée data de la liste ;* *ma_liste*.***replace (*** *data 1*, *data 2* ***)*** *→ remplace data 1 par data 2 ; … et bien d'autres encore … mais,* ***si ces fonctions peuvent faire le travail à votre place, elles ne sont pas toujours autorisées dans les exercices d'évaluation !***
Il est donc nécessaire de savoir le faire par soi même …

=> Voir p.5 (les fonctions) pour le complément sur les listes : le slicing : liste [début : fin]

« Afficher (…) » → `print ( … )`

Afficher du **texte** →
Afficher la **valeur référencée par une variable** →
Effectuer **plusieurs informations à la suite** (texte, variable, …) avec une seule instruction print ( ) →
(séparer chaque information à afficher par une virgule)

```
>>> print("La petite maison")
 La petite maison
>>> val = 3.14
>>> print( val )
 3.14
>>> print("Valeur approchée de Pi : ", val)
 Valeur approchée de Pi :  3.14
```

Si on souhaite utiliser **plusieurs fois l'instruction print (…) mais afficher les informations sur une seule ligne** *(cas des boucles par exemple)* on indique que l'indicateur de fin de ligne **end** aura pour valeur **""** *(pas de fin de ligne)*

```
1 liste = [ 2, 3, 5, 7 ]
2 print("les nombres ", end="")
3
4 for n in liste :
5     print(n, ";" , end="")
6
7 print("sont des nombres premiers.")
```

Console

```
les nombres 2 ;3 ;5 ;7 ;sont des nombres premiers.
```

Mettre en forme l'affichage d'une donnée : **"{ : *code_de_mise_en_forme*} ".format (data)**.

Afficher le "contenu" d'une variable **dans un texte avec une mise en forme** pour soigner la présentation.

```
1 prix = 5.1
2 pourcent = 10
3 print("Le prix est {:>6.02f} € avec {:>2d}% de remise.".format(prix, pourcent) )
4 prix = 123
5 pourcent = 5
6 print("Le prix est {:>6.02f} € avec {:>2d}% de remise.".format(prix, pourcent) )
```

Console

```
Le prix est   5.10 € avec 10% de remise.
Le prix est 123.00 € avec  5% de remise.
```

Affichage de type **str** → {: … s}
Mise au format **int** → {: … d}
Mise au format **float** → {: … f}

(option) **Alignemer** à gauche → {:< … } ou à droite → { :> … }

(option) Indiquer le **nombre total de places à réserver** {:n …}
Forcer l'affichage des zéros non significatifs exemple : 012 plutôt que 12 → {:0n …}

(option) Afficher au format décimal : pour préciser en plus le nombre de chiffres après la virgule et forcer l'affichage des zéros non significatifs exemple : 5.10 € et non 5.1 € → {:.0n f} → n chiffres après la virgule *(le point est compté dans le nombre de caractères au total)*.

*Exemple : 8 caractères au total avec 3 chiffres après le point et forcer l'affichage des zéros non significatifs : " **{:8.03f}** ".format(ma_variable)*.

« Demander à l'utilisateur (…) » → input (…)

La fonction **input (*texte*)** permet d'afficher un *texte* et d'attendre que l'utilisateur ait saisi une information au clavier, dans la console, à la suite du *texte* affiché.

Ici, la variable *saisie* récupère au format str l'information renvoyée par la fonction input.

Cette information peut ensuite être converti en entier **int( )**, si la méthode *saisie*.**isdecimal( )** renvoie la valeur **True**.

```
>>> saisie = input("Entrez un nombre ->")
 Entrez un nombre ->23
>>> saisie
'23'
>>> saisie.isdecimal() # Test avant int( )
True
>>> val = int ( saisie )
>>> print ("La valeur est : ", val )
 La valeur est :  23
```

LES BOUCLES

Boucle « **Pour** *compteur* **dans la liste de valeurs […] »** → `for` *compteur* `in [ … ]`

La boucle **pour** est utilisée lorsqu'on connait à l'avance combien il va y avoir d'itérations. Le ***compteur*** change de valeur à chaque boucle et prend toutes les valeurs de la liste indiquée après **in**.

```
print("Range (3, 9, 2 ) -> ", end="")
for compteur in range(3, 9, 2):
    print( compteur, " ", end="" )

print("\n[ 3, 5, 7 ]      -> ", end="")
for compteur in [ 3, 5, 7 ] :
    print( compteur, " ", end=""  )

print("\nliste = [3,5,7]  -> ", end="")
liste = [ 3, 5, 7 ]
for compteur in liste :
    print ( compteur, " ", end=""  )
```

```
Range (3, 9, 2 ) -> 3  5  7
[ 3, 5, 7 ]      -> 3  5  7
liste = [3,5,7]  -> 3  5  7
```

Note pour la liste de valeurs :
Vous pouvez déclarer la liste de valeurs de différentes manières *(voir code ci-contre →)*

range(*début**, **fin****, *pas** **)** ** → valeurs non obligatoires*
début *vaut 0 par défaut,*
pas *vaut +1 par défaut.*

range(4) → [0, 1, 2, 3]

range (3, 9, 2) → liste de (3 compris) à (9 exclus) , (de +2 en +2) →

Note : Etant donné que str est une liste de caractères, on peut effectuer une boucle for sur une chaîne de caractères.

```
>>> for c in "Maison":
        print(c, ".", end='')

M .a .i .s .o .n .
```

Résumé : 3 méthodes pour utiliser une liste de valeurs dans la boucle for :

range (*[debut ,* **fin [** *, pas* **)**	*Liste de valeurs :* **['a', 'b', 'c']**	**Variable** *de type list*

Savoir faire : créer une liste par compréhension

`liste = [ donnée for compteur in liste ]`

La donnée stockée dans la liste peut être calculée à partir des valeurs que va prendre le compteur dans la liste.

```
>>> # Crée une liste qui contient 15 zéro
>>> liste1 = [ 0 for i in range(15) ]
>>> liste1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> # Crée une liste avec des valeurs calculées
>>> liste2 = [ 2*i for i in range(10) ]
>>> liste2
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Boucle « **Tant que** *condition vraie* **: »** → `while` *condition vraie* **:**

La boucle tant que est utilisée lorsqu'on ne sait pas à l'avance combien il va y avoir d'itérations *(de tours de boucle)*.

Exemple : lorsqu'on demande un nombre à un utilisateur →

```
nombre = None # Initialisation de nombre à 'vide'

while nombre == None :
    saisie = input("Entrez un nombre entier :")
    if saisie.isdecimal() == True :
        # Pour éviter une erreur, la conversion n'est
        # effectuée que si la saisie correspond à un nombre
        nombre = int( saisie )
    else :
        print("Saisie incorrecte, merci de recommencer ...")
```

Note :
On utilise toujours une variable pour tester la condition de fin de boucle.

Ici dans l'exemple, pour pouvoir commencer la boucle **while** , *nombre* est initialisé avec la valeur *None* (vide). Cette valeur n'est modifiée que si la saisie de l'utilisateur est convertible en entier (int). C'est le but du test *.isdecimal ( )*

LES FONCTIONS

def** ma_fonction **(** arguments** **)** -> type du return* **:

> """ docstring → explications sur la finalité de cette fonction """ *(conseillé *)*
>
> *# bloc d'instructions de la fonction indenté*
>
> *return* valeur ***

** → n'est pas indispensable ↔ optionnel*

*** → si nécessaire, selon les besoins dans la fonction*

```
1  from random import randint
2
3  def mini_maxi ( ma_liste:list )->tuple :
4      """
5      Cette fonction parcours les données
6      contenues dans la liste pour déterminer
7      les valeurs minimum et maximum.
8      Reçoit : ma_liste -> la liste des données
9      Renvoie : un tuple ( minimum, maximum )
10     """
11     minimum = ma_liste[0] # Initialiser mini et maxi
12     maximum = ma_liste[0] # avec la première valeur
13
14     for val in ma_liste[1:] :
15         if val < minimum :
16             minimum = val # Mise à jour de mini
17         if maximum < val :
18             maximum = val # Mise à jour de maxi
19     return minimum, maximum
20
21 liste = [ randint(10, 90) for i in range(25) ]
22 mini, maxi = mini_maxi ( liste )
23 print("Minimum : ",mini, " / Maximum : ", maxi)
24
```

Console

```
Minimum :  17  / Maximum :  90
```

Quelques explications :

La fonction **mini_maxi (…)**

Le mot clé **def à la ligne 3** indique à Python que le bloc d'instructions qui va suivre est une fonction.
La ligne 3 se termine par ' : '

Ligne 3 : La fonction reçoit une liste en argument et renvoie un tuple en retour.

Ligne 19 : L'instruction **return minimum, maximum** va mettre fin à la fonction et renvoyer ces deux valeurs au **code d'appel**.

Ligne 22 : "**Appel**" de la fonction **mini_maxi (…)**.
Cette ligne va indiquer à Python qu'il doit exécuter les instructions de la fonction avec la liste passée en argument.

Ligne 22 : les variables mini et maxi vont récupérer les données renvoyées par la fonction mini_maxi (…) lors de l'instruction **return minimum, maximum.**

LIST : Le slicing consiste à :

1 - ne prendre qu'une des données :

1.1 - du début à l'index **n** exclus :
ma_liste **[: n]**
1.2 – de l'index i compris à j exclus :
ma_liste **[i : j]**
1.3 – de **n** inclus à la fin :
ma_liste **[n :]**

2 – faire une copie d'une liste :
copie = ma_liste**[:]**

Les fonctions permettent :

1 – De répartir une tâche complexe en plusieurs tâches simples. Ceci **facilite la lecture du code et simplifie les tests** et la correction des erreurs *(débogage)*

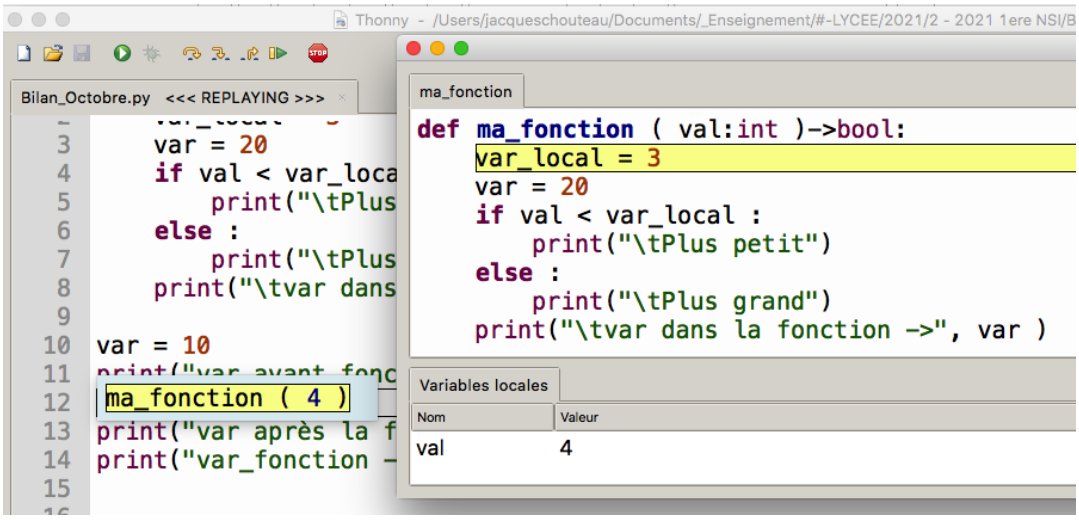
2 – **Eviter de recopier les mêmes lignes de code** si des tâches sont répétitives. Ceci diminue le nombre d'instructions et donc facilite la maintenance du code *(mise à jour)*

3 – Permet de **se répartir le travail** entre plusieurs développeurs.

Quelques points importants :

1- Notions de variable locale / variable globale :

Vous pouvez suivre l'exécution du script par Python en utilisant l'option débogage.



Vous pouvez constater que Thonny **ouvre une nouvelle fenêtre lorsqu'il arrive dans le script d'une fonction**. Ceci montre qu'un fonction a bien son espace personnel de fonctionnement tout en restant connectée au reste du programme.

Les variables val, var_local déclarées dans la fonction n'existent que dans cet espace : **l'espace locale de la fonction**. Les variables sont dites « locales ».

Vous pouvez constater que la variable val, variable locale à la fonction, a bien reçue la valeur 4, passée par « argument » lors de l'appel de la fonction en ligne 12.

La variable var existe dans les deux espaces : la fonction principale (ligne 10) et la fonction ma_fonction ( ) ligne 3 ! **Il y donc un risque de confusion !**
À la ligne 3, lorsque Python exécutera l'instruction var=20 dans le code de la fonction, il va donc créer une nouvelle variable var. Celle-ci sera physiquement indépendante de la variable var de la fonction principale.

Note à ce sujet : *Les variables déclarées à l'extérieur de la fonction sont dites globales et pourraient être utilisées dans la fonction même si ce serait une très mauvaise idée …*
Par principe, une fonction utilise ses propres variables (locales) et les valeurs qu'elle reçoit via les arguments.
Dans certains langages, le recours à des variables globales (extérieurs à la fonction) est considéré comme une erreur (principe des effets de bords).

```
1  def modif ( liste ) :
2      liste[1] = 'B'
3
4  l = ['a', 'b', 'c', 'd' ]
5  modif ( l )
6  print( l )
7
```

Console

```
>>> %Run Bilan_Octobre.py
 ['a', 'B', 'c', 'd']
```

Doc 1 : la fonction a modifié la liste

2 – Attention aux listes passées en argument

Attention, si vous passez une liste en argument à une fonction.
Selon votre algorithme, il se peut que les données dans la liste soient modifiées (doc1) ou pas (doc2) !

Des précautions s'imposent donc …

```
1  def modif ( liste ) :
2      liste = ['A', 'B', 'C', 'D']
3
4  l = ['a', 'b', 'c', 'd' ]
5  modif ( l )
6  print( l )
7
```

Console

```
>>> %Run Bilan_Octobre.py
 ['a', 'b', 'c', 'd']
```

Doc 2 : la fonction n'a pas modifié la liste