Chapitre 3 : Types Des Données / Les Variables

I. Les Types des Données :

Reflexion :

```
print("2")
print(2)
```

On constate qu'au niveau de l'affichage en console les 2 print() affichent le même résultat « 2 », mais en fait les manières de stocker en mémoire par la machine et manipuler ces 2 données sont complétement différentes.
Dans le 1er cas, il s'agit du caractère «2», tout comme les lettres « a » ou « z ».
Dans le second cas, il s'agit du nombre « 2 » qui a une toute autre signification.
Python est capable de reconnaitre chaque donnée entrée et de définir son type il existe 4 types de données

1- Les entiers (Integers) :

Chaque donnée numérique sans virgules on les appelle des entiers en anglais : **Integers** (**int** comme abréviation) qu'ils étaient positifs ou négatifs . Normalement on travaille avec des données numériques à la base décimal mais on peut tout de même représenter les données numériques dans 3 bases : Binaire / Octale / Hexadécimal

Représentation binaire (base 2) : Les nombres ne peuvent être représentés qu'avec les 2 chiffres 0 et 1, (rappelez-vous pour la base 10, on avait 10 chiffres : 0 à 9).

Exemple : Le nombre 1101 en binaire vaut : 1 x 20 + 0 x 21 + 1 x 22 + 1 x 23 = 13 (en base 10)

Représentation octale (base 8) : Les nombres ne peuvent être représentés qu'avec les 8 chiffres 0 à 7

Exemple : Le nombre 227 en octale vaut : 7 x 80 + 2 x 81 + 2 x 82 = 151 (en base 10)

Représentation hexadécimale (base 16) : Les nombres ne peuvent être représentés qu'avec les 16 symboles (0 à 9 ensuite A pour 10, B pour 11, C pour 12, D pour 13, E pour 14 et F pour 15), c'est tout à fait normal de ne représenter 10 à 15 par un seul symbole sinon ça représenterait d'autres nombres et ce serait faux !

Exemple : Le nombre A2C en hexadécimale vaut :

12 x 160 + 2 x 161 + 10 x 162 = 2604 (en base 10) 25

Le langage python reconnaît bien évidement ces 3 représentations, ainsi pour représenter un nombre en binaire, il suffit de le faire précéder par **0b (zéro b)**, par exemple : 0b1101

Pour représenter un nombre en octale, il faut le préfixer par **0o (Zéro o)**, par exemple : 0o227

Pour représenter un nombre en hexadécimale, il faut le préfixer par **0x (Zéro x)**, par exemple : 0xA2C

Voici ce que cela donne avec la fonction print() (Constatez que l'on retrouve les mêmes résultats que plus haut) :

```
print(0b1101)
```

```
Console >_
13
```

```
print(0o227)
```


```
151
```

```
print(0xA2C)
```


```
2604
```

2- Les réels ou flottants (Float) :

Comme le type entier « integer » qu'on vient de voir, il existe dans Python un autre type les nombres réels ou flottants en anglais « **float** » de nombre avec une partie fractionnaire (après la virgule) non vide.

Exemples : 2.5 ; -0.5 ; 256.1789

Important : le nombre 4 tout court est considéré comme entier par Python, en revanche 4.0 est considéré comme un float.

Il faut préciser qu'un « float » se note 2.5 (point) et non 2,5 (la virgule n'est pas admise, elle sert à autre chose dans Python !).

```
print(2.5)
```

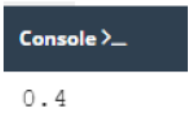

```
2.5
```

```
print(2,5)
```

Dans ce 2ème exemple la virgule « , » signifie que la fonction print() a 2 arguments qui sont les 2 nombres : 2 et 5, d'où l'affichage :

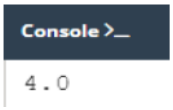
Pour Python le nombre décimal 0.4 peut être écrit .4 (en enlevant le zéro), mais ceci ne marche bien évidemment que pour le zéro :

```
print(.4)
```



De même le nombre 4.0 peut être écrit comme 4. (en enlevant le zéro) et cela ne changera ni son type (float) ni sa valeur :

```
print(4.)
```



Float VS Integer :

Un chose importante à savoir est que le point « . » du décimal est essentiel dans Python pour reconnaître les nombre à virgule flottante, en effet, le nombre 4 tout court est un entier (integer) pour Python, alors que 4.0 est un décimal (float) !

Un autre moyen d'utiliser les « float » est la lettre « e » comme exposant. Ce type de notation est particulièrement intéressant pour les très grands ou très petits nombres réels pour lesquelles la notation scientifique (avec exposant) s'avère intéressante.

Exemple : la vitesse de la lumière est de 300000000 m/s ou 3.108 m/s

En Python on l'écrit comme 3e8 ou 3E8 (il est nécessaire que l'exposant soit un nombre entier !)

```
print(3e8)
print(3E8)
```

```
Console >_
300000000.0
300000000.0
```

3- Chaines de caractères (Strings)

Les chaînes de caractères sont aussi un type Python tout comme integer et float. On les utilise lorsqu'on doit traiter des textes tout simplement.

Par exemple le texte "Je suis une chaîne de caractères" est justement une chaine de caractères (string). Donc en Python tout ce qui est mis entre double quotes (guillemets) représente une chaîne de caractères.

Exemple : On souhaite afficher le texte tel quel avec les **" "** : **I like "Monty Python"**

On peut le faire de 2 manières différentes :

La 1ère consiste à utiliser le fameux caractère d'échappement « **** » avant chacun des guillemets :

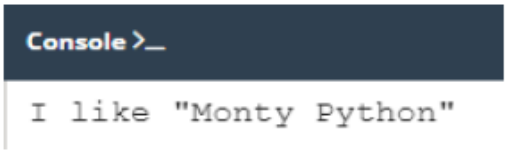
```
print("I like \"Monty Python\"")
```

```
Console >_
I like "Monty Python"
```

La 2ème solution consiste à utiliser l'apostrophe (simple quote) au lieu des doubles quotes pour spécifier une chaine de caractères. Simple quote ou double quote pour les string c'est pareil !

Mais si vous commencez une chaîne avec une apostrophe, vous devez la terminer avec une apostrophe, il faut être cohérent. Dans ce cas, pas besoin du caractère d'échappement « \ » :

```
print('I like "Monty Python"')
```



Remarque : Un chaine de caractères peut être vide, on la note : "" ou ''

4- Les données Booléenes :

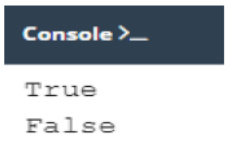
Les valeurs booléennes représentent un autre type Python comme ceux qu'on a vu jusqu'à maintenant mais ne pouvant prendre que 2 valeurs : Vrai (1) et Faux (0) ou en Python **True** et **False** .

Ce type est très important en Python et aussi dans tous les autres langages de programmation.

Exemple :

```
print(10 > 5)
print(10 < 5)
```

Là encore la 1ère expression est vraie donc affichage de « True » et la 2ème est fausse donc affichage de « False » :



II. Exercices

Exercice 1

Quels sont les types des 2 données suivants ?
"Hello ", "007"

Exercice 2

Quels sont les types des littéraux suivants ?
"1.5", 2.0, 528, False

Exercice 3
Quelle est la valeur décimale du nombre binaire suivant ?
1011
Comment faire la conversion en python ?

III. Les opérations et Les opérateurs :

1- Opérateurs de base :

Comme tout langage de programmation qui se respecte, Python reconnaît les opérateurs arithmétiques les plus reconnus :

+ addition, - soustraction, * multiplication, / division, // division entière, % reste de la division euclidienne, ** exposant

Lorsqu'on regroupe des **données** et des **opérateurs** cela forme des **expressions**.

2- Opérateur d'exponentiation ** :

Le signe ** (double étoile) est un opérateur d'exponentiation (puissance). Son argument de gauche est la base, son argument de droite, l'exposant : 23 s'écrit 2 ** 3 en Python.

```
print(2 * 3)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
```

D'où : lorsque les deux arguments de l'opérateur « ** » sont des entiers, le résultat est également un entier ; lorsqu'au moins un des arguments est un float, le résultat est également un float.

3- Opérateur de multiplication * :

```
print(2 * 3)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
```

Son comportement est similaire à l'opérateur d'exponentiation « ** »

4- Opérateur de division / :

6 / 3 – 6 : dividende et 3 : diviseur

print(6 / 3)

print(6 / 3.)

print(6. / 3)

print(6. / 3.)

Nous constatons que le résultat de la division est toujours un float !

Justement, lorsqu'on souhaite que le résultat d'une division soit entier, on fait appel à :

5- Opérateur de division entière // :

print(6 // 4)

print(6. // 4)

print(-6 // 4)

print(6. // -4)

On pourrait croire que -6 // 4 = -1 (car -6 / 4 = -1.5), mais non, -6 // 4 = -2 !

Le résultat de la division des nombres entiers est toujours arrondi à la valeur entière la plus proche qui est inférieure au résultat réel (non arrondi).

C'est très important : l'arrondi se fait toujours sur **le nombre entier le plus petit**.

6- Opérateur : reste de la division (modulo) % :

6 % 4 = 2, car 6 / 4 = 1 et le reste de la division est justement 2 (6 = 4 * 1 + 2).

print(14 % 4)

print(12 % 4.5)

En effet, 14 / 4 = 3 et le reste est 2. 12 / 4.5 = 2 et le reste est 3 (2 * 4.5 = 9.0).

Remarque : Python ne permet pas n'importe quelle division par zéro.

7- Opérateur d'addition + :

print(-4 + 4)

print(-4. + 8)

8- Opérateur de soustraction - les opérateurs unaire et binaire :

L'opérateur « - » peut être utilisé de manière **binaire** (entre 2 opérandes) : 4 – 2 = 2, ou de manière **unaire** pour changer le signe d'un nombre : -2 (l'opposé de 2).

print(-4 - 4)

print(4. - 8)

print(-1.1)

print(+2)

9- Opérateur pour les chaines de caractères :

Concaténation : l'opérateur **« + »** permet de concaténer 2 chaines de caractère

Répétition : L'opérateur « * » permet de répéter la chaine de caractère un nombre de fois

Exemple :

```
>>> print("TDI"*2)
TDITDI
```

Ordre de priorité des opérateurs :

On considère l'exemple suivant où ne met aucune parenthèse. Est-ce que le résultat est 2 + (3*5) = 17 ou (2+3)*5 = 25. 33

print(2 + 3 * 5)

Les opérateurs et leurs liaisons :

A priorité égale, on effectue généralement les calculs de gauche à droite pour tous les opérateurs sauf l'exponentiation « ** ».

print(9 % 6 % 2)

En effet, de gauche à droite : 9 % 6 = 3 et 3 % 2 = 1

Python ne fait pas 6 % 2 = 0 et ensuite on sera embêté avec 9 % 0 !

Pour l'opérateur d'exponentiation, c'est différent :

print(2 ** 2 ** 3)

Python fait le traitement de droite à gauche, d'abord 2 ** 3 = 8, ensuite 2 ** 8 = 256

Et non pas de gauche à droite : 2 ** 2 = 4, ensuite 4 ** 3 = 64

Recapitulation :

Symbole	Opération	Type	Exemple
+	Addition	Entiers, réels, chaines de caractères	6 + 4 → 10 "TDI" + "2020" → "TDI2020"
-	Soustraction	entier, réels	6 – 4 → 2
*	Produit	entier, réels, chaines de caractères	6 * 4 → 24 1.2 * 2 → 2.4 3 * "TDI« → TDITDITDI
**	Puissance	Entiers, réels	12 ** 2 → 144
/	Division	Entiers, réels	6 / 4 = 1.5
//	Division entière	Entiers, réels	6 // 4 → 1
%	Modulo	Entiers, réels	6 % 4 → 2

IV. Priorités des opérations :

Priorité décroissante	Opérateur	Genre
1	+ , -	Unaire
2	**	
3	* , / , %	
4	+ , -	Binaire

+ et - unaires ont la priorité la plus élevée.

Ensuite **, suivi de * , / , et %, puis la priorité la plus faible : + et – binaire.

print(2 * 3 % 5)

En effet, * est prioritaire sur %, donc 2 * 3 = 6 % 5 = 1

Et non pas, 3 % 5 = 3 * 2 = 6 !

Opérateurs et parenthèses

Python permet bien évidement l'utilisation des parenthèses et les sous-expressions entre parenthèses sont prioritaires.

print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)

En effet, 25%13 = 12 + 100 = 112 / 26 (2*13) = 4.307… * 5 = 21.538…// 2 = 10.0

V. Exercices

Exercice 1 :

Essayez de prédire le résultat de chacune des instructions suivantes, puis vérifiez-le dans l'interpréteur Python :

- (1+2)**3
- "Da" * 4
- "Da" + 3
- ("Pa"+"La") * 2
- ("Da"*4) / 2
- 5 / 2
- 5 // 2
- 5 % 2

Exercice 2 :
Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((2 ** 4), (2 * 4.), (2 * 4))
```

Exercice 3 :
Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((-2 / 4), (2 / 4), (2 // 4), (-2 // 4))
```

Exercice 4 :
Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((2 % -4), (2 % 4), (2 ** 3 ** 2))
```

VI. Les variables

Rappel : Une variable est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom, alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.

Avec Python on est capable de stocker des données contenant des valeurs numériques et textuelles sur lesquelles on peut effectuer des opérations arithmétiques.

Tout de même , on les stocke dans des variables mais **En Python,** la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps. on a pas besoin de préciser le type d'une variable car la variable en python prend automatiquement le type de la valeur qu'on va stocker dans cette variables

➔ En Python une variable doit avoir :
- Un nom ;
- Une valeur (le contenu du conteneur).

Exemple :

```
a = 2
print(a)
```

Ligne 1. Dans cet exemple, nous avons déclaré, puis initialisé la variable x avec la valeur 2. Notez bien qu'en réalité, il s'est passé plusieurs choses :

- Python a « deviné » que la variable était un entier. On dit que Python est un langage au typage dynamique.
- Python a alloué (réservé) l'espace en mémoire pour y accueillir un entier. Chaque type de variable prend plus ou moins d'espace en mémoire. Python a aussi fait en sorte qu'on puisse retrouver la variable sous le nom x.
- Enfin, Python a assigné la valeur 2 à la variable x.

Opérateur d'affectation :

Pour affecter ou "assigner" une valeur à une variable, nous allons utiliser un opérateur qu'on appelle opérateur d'affectation ou d'assignation et qui est

représenté par le signe =. Attention, le signe = ne signifie pas en informatique l'égalité d'un point de vue mathématique : c'est un opérateur d'affectation.

Le signe = ne sert pas à dire que la valeur est égale au nom de variable ou que la variable "vaut" cette valeur, il indique simplement qu'on affecte ou qu'on stocke une certaine valeur dans un conteneur.

Mais il existe d'autres types d'affectation qui nous permet d'affecter directement et automatiquement le résultat d'une opération

Opérateur	Exemple	Equivalent à	Description
=	x = 1	x = 1	Affecte 1 à la variable x
+=	x += 1	x = x + 1	Ajoute 1 à la dernière valeur connue de x et affecte la nouvelle valeur (l'ancienne + 1) à x
-=	x -= 1	x = x – 1	Enlève 1 à la dernière valeur connue de x et affecte la nouvelle valeur à x
*=	x *= 2	x = x * 2	Multiplie par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
/=	x /= 2	x = x / 2	Divise par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
%=	x %= 2	x = x % 2	Calcule le reste de la division entière de x par 2 et affecte ce reste à x
//=	x //= 2	x = x // 2	Calcule le résultat entier de la division de x par 2 et affecte ce résultat à x
**=	x **= 4	x = x ** 4	Elève x à la puissance 4 et affecte la nouvelle valeur dans x

Type des Variables :

Puisque Pyhton est un language de typage dynamique alors les types des variables sont de meme les types des données qu'on y stocke est alors les types des variables sont aussi 4 :

- Les entiers (*integer* ou *int*),
- Les nombres décimaux que nous appellerons *floats*
- Les chaînes de caractères (*string* ou *str*).
- Les booléens qui accepte deux valeurs (true ou False)

Exemple :

```
# declaration d'un entier
x = 3

#declaration d'un nombre decimale
y = 3.14

#declaration d'une chaine de caracteres
z1 = "TDI 2020"
z2 = 'TDI 2020'

#declaration d'un boolean
t1 = True
t2 = False
```

Les règles de nommage :

1- Le nom des variables en Python peut être constitué de lettres minuscules (a à z), de lettres majuscules (A à Z), de nombres (0 à 9) ou du caractère souligné (_). Vous ne pouvez pas utiliser d'espace dans un nom de variable.

2- Un nom de variable ne doit pas débuter par un chiffre.

3- De plus, il faut absolument éviter d'utiliser un mot « réservé » par Python comme nom de variable (par exemple : print, range, for, from, etc.).

4- Python est sensible à la casse :

   a. Ce qui signifie que les variables TesT, test ou TEST sont différentes.

Affichage des variables :

La fonction print() peut également afficher le contenu d'une variable quel que soit son type. Voici des exemples :

```
#Exemple 1
var = 3
print(var)

#Exemple 2
age = 32
name1 = 'zaid'
print(name1, "a",age,"ans")

#Exemple 3
note = 13.1
name2 = 'Riyad'
print("{0} a eu {1:.2f} en mathematique".format(name2,note))
```

Exercice d'Apprentissage :

Il était une fois à Appleland, Jean avait trois pommes, Marie cinq pommes et Adam six pommes. Ils étaient tous très heureux et ont vécu longtemps. Fin de l'histoire.

1- Créer les variables : john, mary, et adam ;
2- Attribuer des valeurs aux variables. Les valeurs doivent être égales au nombre de fruits possédés par Jean, Marie et Adam respectivement ;
3- Après avoir enregistré les nombres dans les variables, imprimez les variables sur une ligne et séparez-les par une virgule ;
4- Créez maintenant une nouvelle variable appelée totalPommes, égale à l'addition des trois anciennes variables.
5- Imprimez la valeur stockée dans totalApples sur la console ;
6- Expérimentez votre code : créez de nouvelles variables, attribuez-leur différentes valeurs et effectuez diverses opérations arithmétiques sur elles (par exemple, +, -, *, /, //, etc.).

VII. La fonction : Type() :

La fonction type retourne le type de données d'un objet quelconque.

Si vous ne vous souvenez plus du type d'une variable, utilisez la fonction **type()** qui vous le rappellera.

```
x = 3
y = 4.5
t = "TDI 2020"
u = False

print("Le type de x est : ", type(x))
print("Le type de y est : ", type(y))
print("Le type de t est : ", type(t))
print("Le type de u est : ", type(u))
```

Le type de x est : <class 'int'>
Le type de y est : <class 'float'>
Le type de t est : <class 'str'>
Le type de u est : <class 'bool'>

La conversion des types :

En programmation, on est souvent amené à convertir les types, c'est-à-dire passer d'un type numérique à une chaîne de caractères ou vice-versa. En Python, rien de plus simple avec les fonctions int(), float() et str().

Voici un exemple d'utilisation :

```
x = "3.14"
y = float(x)

print("La valeur de x est : ",x)
print("La valeur de y est : ",y)

print("Le type de x est : ",type(x))
print("Le type de y est : ",type(y))
```

La valeur de x est : 3.14
La valeur de y est : 3.14
Le type de x est : <class 'str'>
Le type de y est : <class 'float'>