ISN – Informatique et Sciences du Numérique

LANGAGE PYTHON : TYPES, VARIABLES ET OPERATEURS

1 – TYPE INT (INTEGER : NOMBRES ENTIERS)

Par défaut, les **entiers** sont des nombres décimaux, mais il est possible d'utiliser également la base binaire ou hexadécimale.

Exemple 1 : Entiers

```
>>> 2013              # décimal
2013
>>> 0b11111011101     # binaire
2013
>>> 0x7DD             # hexadécimal
2013
```

Les principales opérations arithmétiques :

Exemple 2 : Opérations arithmétiques sur les entiers

```
>>> 50 + 3           # addition
53
>>> 50 - 3           # soustraction
47
>>> 50 * 3           # multiplication
150
>>> 50 / 3           # division
16.666666666666668
>>> 50 // 3          # division entière
16
>>> 50 % 3           # modulo
2
```

2 – TYPE FLOAT (FLOTTANT OU A VIRGULE FLOTTANTE)

Une donnée de type **float** ou **réelle** est notée avec un point décimal ou en notation exponentielle :

Exemple 3 : Flottants

```
>>> 4.215
4.215
>>> .0087
0.0087
>>> 8e9
8000000000.0
>>> 1.025e38
1.025e+38
```

Les flottants **supportent les mêmes opérations** que les entiers. Ils ont une **précision finie**.

L'importation du **module math** permet l'utilisation de **fonctions mathématiques usuelles**.

Exemple 4 : Fonctions mathématiques usuelles

```
>>> import math
>>> dir(math)
['__doc__','__name__','__package__','acos','acosh','asin','asinh','atan','atan2','atanh',
'ceil','copysign','cos','cosh','degrees','e','erf','erfc','exp','expm1','fabs','factorial'
,
'floor','fmod','frexp','fsum','gamma','hypot','isfinite','isinf','isnan','ldexp','lgamma',
'log','log10','log1p','modf','pi','pow','radians','sin','sinh','sqrt','tan','tanh','trunc'
]
>>> math.sin(math.pi/4)      # sin(pi/4)
0.7.71067811865475
>>> math.degrees(math.pi)        # pi en degrés
180.0
>>> math.sqrt(2)         # racine carrée de 2
1.4142125623730951
```

3 – TYPE BOOL (BOOLEEN)

Les données du type **bool** ne présentent que deux valeurs : **False** et **True**. Les opérations logiques et comparaison **sont évaluées** et le résultat est un **booléen**.

Exemple 5 : Opérateurs de comparaison

```
1>>> 2 < 8       # strictement inférieur
True
>>> 2 <= 8       # inférieur ou égal
True
>>> 2 == 8       # égal
False
>>> 2 > 8        # strictement supérieur
False
>>> 2 >= 8       # supérieur ou égal
False
>>> 2 != 8       # différent
True
```

Exemple 6 : Opérateurs logiques

```
>>> (3 == 3) or (9 > 24)     # OU logique
True
>>> (9 > 24) and (3 == 3)        # ET logique
False
>>> not(3 == 3)          # NON logique
False
```

4 – TYPE STR (STRING OU CHAINE DE CARACTERE)

Une donnée de type **str** représente une séquence constituée de caractères.

Exemple 7 : Représentation d'une chaîne de caractères

```
>>> "Dupont"         # utilisation des guillemets
'Dupont'
>>> 'Pierre'         # utilisation des apostrophes
'Pierre'
```

Pour une chaîne de caractères avec apostrophes, il faut utiliser la **séquence d'échappement **.

**Exemple 8 : Séquence d'échappement **

```
>>> 'Aujourd'hui'
  File "<intercative input>", line 1
    'Aujourd'hui'
¨              ^
SyntaxError: invalid syntax
>>> 'Aujourd\'hui'       # utilisation de la séquence d'échappement
"Aujourd'hui"
```

Pour un **saut à la ligne** il faut utiliser la **séquence d'échappement \n** ou la forme **multi-lignes** avec **triples guillemets**.

Exemple 9 : Saut à la ligne

```
>>> chaine = 'Dupont\nPierre'    # séquence d'échappement \n
>>> print(chaine)
Dupont
Pierre
>>> chaine = """Dupont
... Pierre"""        # Forme multi-lignes
>>> print(chaine)
Dupont
Pierre
```

Opérations sur les chaines de caractères :

Exemple 10 : Opérations sur les chaînes de caractères

```
>>> 'Dupont'+' '+'Pierre'        # concaténation de chaînes de caractères
'Dupont Pierre'
>>> chaine = 'Dupont Pierre'
>>> len(chaine)          # longueur d'une chaîne de caractères
13
>>> chaine = 'Ha ! '
>>> chaine * 3           # répétition
>>> print(chaine)
'Ha ! Ha ! Ha ! '
```

Exemple 11 : Indexage

```
>>> chaine = 'Dupont Pierre'
>>> print(chaine[0])     # premier caractère
D
>>> print(chaine[-1])        # dernier caractère
e
>>> print(chaine[2:6])       # du 3ième au 7ième caractère
pont
```

Il n'est pas possible de réaliser des opérations arithmétiques sur des chaînes de caractères. La fonction **float()** permet de convertir **un type str en type float** et la fonction **int()** permet de convertir **un type str en type int**.

Exemple 12 : fonctions int() et float()

```
>>> '17.45' + 2      # opération impossible
Traceback (most recent call last):
 File "<interactive input>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> float('17.45') + 2   # utilisation de la fonction float()
19.45
>>> int('3') * 2     # utilisation de la fonction int()
6
```

5 – TYPE LIST (LISTE)

Une liste est une structure de données. Le premier élément d'une liste possède **l'indice** (ou **l'index**) **0**. Une liste peut être constituée **d'éléments types différents**.

Exemple 13 : type list

```
>>> Donnees = ['Dupont','Pierre',17,1.75,72.5]    # liste constituée de str, int et float
>>> print(Donnees)
['Dupont','Pierre',17,1.75,72.5]
>>> print('Nom : ',Donnees[0])                 # premier élément indice 0
Nom : Dupont
>>> print('Age : ',Donnees[2])                 # troisième élément indice 2
Age : 17
>>> print('Taille : ',Donnees[3])              # quatrième élément indice 3
Taille : 1.75
```

Il est possible de créer des **listes à 2 dimensions** (équivalentes à des tableaux).

Exemple 14 : liste à 2 dimensions

```
>>> liste = [[0,1,2],[3,4,5],[6,7,8]]         # liste à 2 dimensions
>>> print(liste)
[[0,1,2],[3,4,5],[6,7,8]]
>>> print(liste[0])              # éléments de la 1ère ligne
[0,1,2]
>>> print(liste[1][2])               # éléments de la 2nde ligne et 3ième colonne
5
```

6 – TYPE DICT (DICTIONNAIRE)

Un dictionnaire permet de **stocker des données sous la forme (clé ; valeur)**. Une clé est unique et n'est pas nécessairement un entier.

Exemple 15 : type dict

```
>>> moyenne = {'Math':14,'Anglais':12.5,'Français':13}
>>> print(moyenne)                # tout le dictionnaire
{'Anglais':12.5,'Français':13,'Math':14}
>>> print(moyenne['Math'])        # la valeur qui a pour clé « Math »
14
>>> moyenne['Anglais'] = 16       # nouvelle affectation
>>> print(moyenne)                # tout le dictionnaire
{'Anglais':16,'Français':13,'Math':14}
```

7 – AUTRES TYPES

Il en existe bien d'autres types :

- **long** : nombres entiers de longueur quelconque (**4284961775562012536954159102L**) ;
- **complex** : nombres complexes (**1 + 2.5 j**) ;
- **tuple** : structure de données ;
- **file** : fichiers ;
- ...

8 – VARIABLES

Une variable est un **espace mémoire** dans lequel il est possible de **stocker une valeur** (une donnée). Il s'agit donc d'un **identifiant associé à une valeur**.

La notion de variable n'existe pas dans le langage Python. On parle plutôt de **référence d'objet**. Il s'agit donc d'une **référence d'objet située à une adresse mémoire**.

On **affecte une variable** par une valeur en utilisant le signe **=**. Dans une affectation, le membre de gauche reçoit le membre de droite.

Exemple 16 : affectations simples de variables

```
>>> a = 2            # la variable a reçoit la valeur 2
>>> b = math.sqrt(2) # la variable b reçoit la valeur racine carrée 2
>>> c = a*b          # la variable c reçoit la valeur de a fois la valeur de b
>>> print(c)
2.8284271247461903
```

Outre l'affectation simple, on peut aussi utiliser les formes suivantes :

Exemple 17 : autres formes d'affectations de variables

```
>>> a = 3                # affectation simple
>>> print(a)
3
>>> a += 3               # affectation augmentée a = a + 3
>>> print(a)
6
>>> a = b = 7            # affectations multiples
>>> print(a)
7
>>> print(b)
7
>>> a,b = 2.7,5.1            # affectation parallèle de séquences : tuple
>>> print(a)
2.7
>>> print(b)
5.1
>>> a,b,c = ['A','B','C']        # affectation parallèle de séquences : liste
>>> print(a)
A
>>> print(b)
B
>>> print(c)
C
```