RAPPELS PYTHON – Généralités

Types de base :

- **int** : nombre entier relatif (integer) ; intervalle illimité (en Python)
- **float** : nombre en virgule flottante = nombre décimal ; intervalle : de $-1{,}7 \times 10^{308}$ à $+1{,}7 \times 10^{308}$.
- **str** : chaine de caractères (string)
- **bool** : booléen : peut prendre la valeur **True** ou **False**.

Affectation de variable : (l'affectation définit le type de la variable)
*Syntaxe :***nomdevariable = valeur**
Exemples : **n = 2** (nombre entier) ; **x = 3.2** (nombre en virgule flottante) ; **mot = 'test'** (chaine de caractère)
Affectations multiples : **a, b = 2, 3** équivalent à **a = 2** et **b = 3** ; **a = b = 5** équivalent à **a = 5** et **b = 5**
Incrémentation : x += 5 équivalent à x = x + 5
Décrémentation : x –= 5 équivaut à x = x - 5

Affichage d'une variable : print(nomdevariable)

Entrée d'une variable au clavier : nomdevariable = input('Texte :')
Attention, la variable renvoyée par la fonction input() est forcément une chaine de caractère. Si on veut entrer une valeur numérique, utiliser :
nomdevariable = int(input('Texte :')) pour un nombre entier
nomdevariable = float(input('Texte :')) pour un nombre en virgule flottante

Instructions conditionnelles :
Structure générale :
if (condition) :
    instructions exécutées si la condition est vraie (True)
else :
    instructions exécutées si la condition est fausse (False)
Si on veut « chainer » plusieurs tests, on peut utiliser elif :
if (condition1) :
    instructions exécutées si la condition1 est vraie (True)
elif (condition2) :
    instructions exécutées si la condition2 est vraie (True)
elif (condition3) :
    instructions exécutées si la condition3 est vraie (True)
…
else :
    instructions exécutées si toutes les conditions précédentes sont fausses (False)
Remarque : les blocs elif ou else peuvent être omis
Conditions :
a == b : True si a = b, False sinon
a != b : True si a ≠ b, False sinon
a < b : True si a < b, False sinon
a > b : True si a > b, False sinon
a <= b : True si a ≤ b, False sinon
a >= b : True si a ≥ b, False sinon
Conditions multiples :
(a < b) and (c > d) : True si a < b ET si c > d, False sinon
(a < b) or (c > d) : True si a < b OU si c > d, False sinon
Not (a < b) : True si a ≥ b, False sinon

Boucles inconditionnelles :
for i in range(a, b, c) :
    Instructions
Remarques : i doit être un nombre entier. La boucle sera parcourue avec une valeur de i allant de a à (b-1) par pas de c (ce qui veut dire que i est augmenté de c à chaque tour de boucle).

For i in range(a, b) :
    Instructions
Si c est omis, alors i sera augmenté de 1 à chaque tour de boucle.

For i in range(b) :
    Instructions
S'il n'y a qu'un seul paramètre dans **range()**, Python considère que a = 0 et i va donc varier de 0 à (b-1).

Ruptures de séquences :
```
for x in range(0, 10) :
    if x == 5 :
        break
    print(x)
print('Fin')
```
Le programme précédent affiche : 0, 1, 2, 3, 4, Fin : **break** permet de sortir de la boucle.

```
for x in range(0, 10) :
    if x == 5 :
        continue
    print(x)
print('Fin')
```
Le programme précédent affiche : 0, 1, 2, 3, 4, 6, 7, 8, 9, Fin : **continue** permet de sauter une valeur de i.

Boucles conditionnelles :
while (condition) :
    instructions exécutées tant que la condition est vraie (True)

Importation de bibliothèques :
1ère méthode : **import math**
    On importe toutes les fonctions de la bibliothèque « math » et on doit utiliser le préfixe **math.**
    Exemple d'utilisation : **a = math.sqrt(2)**

2ème méthode : **from math import sqrt, log, sin**
    Ici on importe seulement quelques fonctions. On n'a pas besoin d'utiliser le préfixe **math.**
    Exemple d'utilisation : **a = sqrt(2)**

3ème méthode : **from math import ***
    On importe toutes les fonctions de la bibliothèque « math » et on n'a pas besoin du préfixe **math.**
    Exemple d'utilisation : **a = sqrt(2)**

RAPPELS PYTHON – Chaînes de caractères

Une chaîne de caractère est une variable de type **string**.

Définition d'une variable de type string :

```
chaine = 'Bonjour'   ou :   chaine = "Bonjour"
```

Concaténation de chaînes :

```
chaine1 = 'Bonjour, '
chaine2 = 'les amis !'
chaine = chaine1 + chaine2          chaine     renvoie : 'Bonjour, les amis !'
```

Répétition de chaînes :

```
chaine = 'abc'
10 * chaine      renvoie :      'abcabcabcabcabcabcabcabcabc'
'az' * 5         renvoie :      'azazazazaz'
```

Comparaison de chaînes :

```
chaine1 = 'abc'
chaine2 = 'def'
chaine1 < chaine 2          renvoie : True
```

Les chaines sont classées par ordre alphabétiques, plus exactement par le code ASCII de leurs caractères. (Attention aux majuscules et aux minuscules. On a, par exemple, `'Z' < 'a'`)

Longueur d'une chaîne de caractère :

`len(chaine)` : renvoie la longueur de la chaîne, en nombre de caractères.

Exemple : `chaine = 'Bonjour'` `print(len(chaine))` Valeur renvoyée : 7

Extraction d'un caractère d'une chaîne :

`chaine[n]` : renvoie le caractère se trouvant à la position n *(Attention : le premier caractère de la chaîne est à la position 0 !)*

Exemple : `chaine = 'Bonjour'` `chaine[3]` Valeur renvoyée : `'j'`

Découpage de chaîne : (« slices »)

```
chaine = 'Bonjour'
```

B	o	n	j	o	u	r
0	1	2	3	4	5	6	7

```
chaine[ 2 : 5 ]     ⇒     'njo'
chaine[ : 4 ]       ⇒     'Bonj'
chaine[ 3 : ]       ⇒     'jour'
```

Quelques méthodes de la classe string :

`nomdelachaine.lower()` : convertit la chaîne en minuscules
`nomdelachaine.upper()` : convertit la chaîne en majuscules
`nomdelachaine.capitalize()` : convertit la première lettre de la chaîne en majuscule
`nomdelachaine.swapcase()` : inverse les majuscules et les minuscules
`nomdelachaine.strip()` : enlève les éventuels espaces devant et derrière la chaîne

Pour d'autres méthodes : https://docs.python.org/fr/3/library/stdtypes.html#string-methods

Pour aller plus loin :
D'autres instructions disponibles dans la librairie `string` : https://docs.python.org/fr/3/library/string.html

Code ASCII :

Chaque caractère est défini par son code ASCII (valeur entre 0 et 255) (cf. Tableau suivant).

`ord(caractère)` : donne le code ASCII du caractère. *Exemple :* **`ord('A')`** renvoie la valeur 65.

`chr(n)` : renvoie le caractère de code ASCII n. *Exemple :* **`chr(65)`** renvoie **`'A'`**.

Valeurs utiles :

- Les chiffres de '0' à '9' ont des codes ASCII allant de 48 à 57.
- Les lettres majuscules de 'A' à 'Z' ont des codes ASCII allant de 65 à 90.
- Les lettres minuscules de 'a' à 'z' ont des codes ASCII allant de 97 à 122.

On remarque ainsi que partant d'une lettre majuscule, il suffit d'ajouter 32 à son code ASCII pour trouver la lettre minuscule correspondante.

Table des codes ASCII :

0 NUL	32 espace	64 @	96 `
1 SOH	33 !	65 A	97 a
2 STX	34 "	66 B	98 b
3 ETX	35 #	67 C	99 c
4 EOT	36 $	68 D	100 d
5 ENQ	37 %	69 E	101 e
6 ACK	38 &	70 F	102 f
7 BEL	39 '	71 G	103 g
8 BS	40 (	72 H	104 h
9 HT	41)	73 I	105 i
10 LF	42 *	74 J	106 j
11 VT	43 +	75 K	107 k
12 FF	44 ,	76 L	108 l
13 CR	45 -	77 M	109 m
14 SO	46 .	78 N	110 n
15 SI	47 /	79 O	111 o
16 SLE	48 0	80 P	112 p
17 CS1	49 1	81 Q	113 q
18 DC2	50 2	82 R	114 r
19 DC3	51 3	83 S	115 s
20 DC4	52 4	84 T	116 t
21 NAK	53 5	85 U	117 u
22 SYN	54 6	86 V	118 v
23 ETB	55 7	87 W	119 w
24 CAN	56 8	88 X	120 x
25 EM	57 9	89 Y	121 y
26 SIB	58 :	90 Z	122 z
27 ESC	59 ;	91 [	123 {
28 FS	60 <	92 \	124 |
29 GS	61 =	93]	125 }
30 RS	62 >	94 ^	126 ~
31 US	63 ?	95 _	127 ■

(Source : http://sebsauvage.net/comprendre/ascii/)

Les codes de 0 à 31, ainsi que le code 127 sont des codes de contrôle. Ils ont une fonction spéciale et nous n'aurons en règle générale pas besoin de les utiliser.

Pour aller plus loin :

- Page Wikipedia : https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange
- Vidéo « Comment lire du binaire » : https://www.youtube.com/watch?v=wCQSIub_g7M

Une liste est un objet composite de type list.

Définition d'une variable de type list :

```
liste = [ 1, 2, 3.5, 'abc', 5 ]
```

Concaténation de listes :

```
liste1 = [ 1, 2, 3 ]
liste2 = [ 4, 5, 6 ]
liste = liste1 + liste2        renvoie : [ 1, 2, 3, 4, 5, 6 ]
```

Répétition de listes :

```
liste = [ 1, 2 ]
3 * liste       renvoie :    [ 1, 2, 1, 2, 1, 2 ]
liste * 3       renvoie :    [ 1, 2, 1, 2, 1, 2 ]
```

Longueur d'une liste :

len(liste) : renvoie la longueur de la liste, en nombre d'éléments.

Exemple : `liste = [ 1, 2, 3.5 ]` `print(len(liste))` Valeur renvoyée : 3

Extraction d'un élément d'une liste :

liste[n] : renvoie l'élément se trouvant à la position n *(Attention : le premier élément de la liste est à la position 0 !)*

Exemple : `liste = [ 1, 2, 3.5 ]` `liste[1]` Valeur renvoyée : **2**

Découpage de liste : (« slices »)

```
liste = [ 1, 2, 3.5, 'abc', 5 ]
liste[ 1 : 3 ] ⇒    [ 2, 3.5 ]
liste[ : 4 ]   ⇒    [ 1, 2, 3.5, 'abc' ]
liste[ 3 : ]   ⇒    [ 'abc', 5 ]
```

Quelques méthodes de la classe list :

nomdelaliste.append(x) : rajoute l'élément **x** à la fin de la liste
nomdelaliste.insert(i, x) : insère l'élément **x** à la position **i** de la liste
nomdelaliste.pop(i) : enlève l'élément situé à la position **i** de la liste et le renvoie en valeur de retour
nomdelaliste.pop() : enlève de la liste l'élément situé à la fin de la liste et le renvoie en valeur de retour
nomdelaliste.reverse() : inverse l'ordre des éléments de la liste
nomdelaliste.sort() : trie dans l'ordre les éléments de la liste

Attention : Toutes ces méthodes modifient la liste concernée, contrairement aux méthodes agissant sur la classe str.

Exemple :

```
chaine = 'Bonjour'
chaine.upper()      renvoie :    'BONJOUR'
chaine              renvoie :    'bonjour'
```

La chaine de caractère du départ n'a pas été modifiée par la méthode.

```
liste = [ 9, 8, 5, 1, 2, 3 ]
liste.sort()
liste               renvoie :    [1, 2, 3, 5, 8, 9]
```

La liste du départ a été modifiée par la méthode.

Savoir si un élément est dans une liste :

```
liste = [ 1, 2, 3.5, 'abc', 5 ]
3.5 in liste          renvoie : True
3.5 not in liste      renvoie : False
666 in liste          renvoie : False
666 not in liste      renvie : True
```

Liste de listes :

Une liste peut contenir n'importe quel type d'objet, notamment des objets de type list. On peut donc réaliser des listes imbriquées. Cette fonctionnalité permet notamment de réaliser des tableaux.

Exemple :

Soit le tableau suivant :

	x=0	x=1	x=2	x=3
y=0	1	2	3	4
y=1	5	6	7	8
y=2	9	10	11	12
y=3	13	14	15	16

On peut représenter ce tableau avec la liste suivante :

```
tableau = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
```

`tableau[2]` renvoie : `[9, 10, 11, 12]` ce qui correspond à la troisième ligne du tableau (y=2)

`tableau[2][1]` renvoie : `10` ce qui correspond à la deuxième colonne (x=1) de la troisième ligne (y=2)

En règle générale, si on veut accéder à l'élément du tableau se trouvant à la ligne y et à la colonne x, on peut utiliser : `tableau[y][x]`

Remarque : on peut aussi faire une liste de listes de listes, ce qui permet de faire des tableaux 3D. On peut ainsi réaliser des tableaux à plus de trois dimensions, mais cela devient difficile de se représenter le résultat.

Compréhensions de listes :

Les compréhensions de listes sont des méthodes permettant de réaliser des listes de façon très compacte.

Exemple n°1 :

```
carres = [ ]
for i in range(10):
    carres.append(i*i)
print(carres)          renvoie : [ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 ]

carres = [ x*x for x in range(10) ]      renvoie la même chose
```

Exemple n°2 :

```
impairs = [ ]
for i in range(20):
    if ( i % 2 ) != 0:
        impairs.append(i)
print(impairs) renvoie : [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 ]

impairs = [ i for i in range(20) if ( i % 2 ) != 0 ]   renvoie la même chose
```

Pour aller plus loin : https://docs.python.org/fr/3/library/stdtypes.html#sequence-types-list-tuple-range