Types de données en Python

I - Types natifs (https://docs.python.org/fr/3.7/library/stdtypes.html)

1. Objet de type `int` (https://docs.python.org/fr/3.7/library/functions.html#int)

Les objets de type `int` (https://docs.python.org/fr/3.7/library/functions.html#int) sont les entiers relatifs en mathématiques :

In [1]:
```
type(-5)
```
Out[1]:
```
int
```

In [2]:
```
type(20000000000000000)
```
Out[2]:
```
int
```

Ces objets n'ont pas de restriction de taille (à part celle de la mémoire allouée par la machine). Les opérations les plus courantes sont les suivantes :

In [3]:
```
5+6*7-(-2)
```
Out[3]:
```
49
```

In [4]:
```
# Puissance
5**7
```
Out[4]:
```
78125
```

L'opérateur `//` donne le quotient de la division entière (https://docs.python.org/fr/3.7/glossary.html#term-floor-division).

In [5]:
```
28//5
```
Out[5]:
```
5
```

L'opérateur modulo `%` donne le reste de la division entière.

In [6]:
```
28%5
```
Out[6]:
```
3
```

2. Objet de type `float` (https://docs.python.org/fr/3.7/library/functions.html#float)

In [7]:
```
type(8/2)
```
Out[7]:
```
float
```

L'opération `/` retourne en objet de type `float` (https://docs.python.org/fr/3.7/library/functions.html#float) : nombre décimal en mathématiques. On dit également nombre à virgule flottante.

In [8]:
```
# On peut prendre la partie entière d'un nombre décimal
int(3.5)
```
Out[8]:
```
3
```

In [9]:
```
1.7e5
```
Out[9]:
```
170000.0
```

In [10]:

```
from math import sqrt
sqrt(16)
```

Out[10]:

```
4.0
```

3. Le transtypage implicite

Python permet de faire des calculs avec des nombres à virgule flottante. Si on mélange dans un calcul des nombres entiers et des nombres à virgule flottante dans un calcul python fait la conversion du nombre entier en nombre à virgule flottante. On peut parler de transtypage (Python change le type de la variable de int à float) implicite (Python le fait sans rien dire)

In [11]:

```
4+1.0
```

Out[11]:

```
5.0
```

In [12]:

```
2*3.5
```

Out[12]:

```
7.0
```

Remarque : On peut explicitement demander [`le transtypage`] à Python :

In [ ]:

```
int(7.0)
```

Attention, dans certains cas une autre opération est aussi réalisée :

In [13]:

```
int(1.5)
```

Out[13]:

```
1
```

4. Objet de type `bool` (https://docs.python.org/fr/3.7/library/stdtypes.html#truth-value-testing)

Un booléen peut prendre deux valeurs : `True` et `False` . Il est le résultat d'une opération logique :

In [14]:

```
2 < 3
```

Out[14]:

```
True
```

In [15]:

```
3 > 4
```

Out[15]:

```
False
```

In [16]:

```
2 == 2.0
```

Out[16]:

```
True
```

In [17]:

```
False or True
```

Out[17]:

```
True
```

In [18]:

```
False and True
```

Out[18]:

```
False
```

In [19]:

```
0 != 1
```

Out[19]:

```
True
```

Remarque importante : Vous avez du, où vous verrez prochainement comment les nombres décimaux sont représentés machine. En particulier, le nombre décimal 0,1 n'est pas réprésenté de manière exacte en machine. En effet :

In [20]:

```
0.1*3
```

Out[20]:

```
0.30000000000000004
```

Et donc :

In [21]:

```
0.1*3 == 0.3
```

Out[21]:

```
False
```

In [25]:

```
0.1*3 > 0.3
```

Out[25]:

```
True
```

Retenez qu'il ne faut **jamais** utiliser l'opérateur `==` pour les nombres décimaux en Python.

II - Type Séquence de Texte `str` (https://docs.python.org/fr/3.7/library/stdtypes.html#text-sequence-type-str)

1. Quelques règles pour commencer

En plus des nombres, Python est capable de manipuler du texte. Comme un texte est composé de lettres (ou plus généralement de caractère), on parle de **chaînes de caractères**.

Les chaines de caractère peuvent être entourées de guillemets simples `'texte'` ou de guillemets doubles `"texte"` avec un résultat identique, à quelques petits détails prêts.

In [1]:

```
print('Des oeufs brouillés')  # Des guillemets simples
```

```
Des oeufs brouillés
```

In [2]:

```
print('l\'oeuf ou la poule')  # On utilise \' pour afficher une apostrophe
```

```
l'oeuf ou la poule
```

In [3]:

```
print("l'oeuf ou la poule")  # ... ou on utilise les guillemets doubles
```

```
l'oeuf ou la poule
```

In [4]:

```
print('"oui," dis je')
```

```
"oui," dis je
```

Le caractère protégé `\n` (on lit aussi parfois caractère échappé) sert à indiquer à indiquer dans une chaine de caractère un passage à la ligne.

In [5]:

```
print("Première ligne.\nSeconde ligne.")
```

```
Première ligne.
Seconde ligne.
```

Les chaines de caractères peuvent s'étendre sur **plusieurs lignes** en étant délimitées par des de triples guillemets (simples ou doubles). Cela permet d'avoir tout un texte facilement.

Il n'y a alors pas besoin d'indiquer le caractère `\n` pour le passage à la ligne.

In [6]:

```
print("""Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to""")
```

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

2. Concaténation

Les chaînes de caractères peuvent être concaténées, c'est à dire mises bout à bout pour créer une nouvelle chaine. L'opérateur pour effectuer cette opération est le signe `+` .

Une chaîne de caractère peut être répétée plusieurs fois en utilisant l'opérateur `*` avec le nombre de répétitions.

In [7]:

```
# Mettre bout à bout deux chaines
chaine = 'Py' + 'thon'
print(chaine)
```

```
Python
```

In [8]:

```
# Multiplier des chaines
chanson = (2 * "pou" +'pidou\n')*3
print(chanson)
```

```
poupoupidou
poupoupidou
poupoupidou
```

3. Accès aux éléments d'une chaîne de caractères

In [9]:

```
chaine = "ma belle chaîne"
print("Première lettre : ", chaine[0])
```

```
Première lettre :  m
```

In [10]:

```
print("Dernière lettre : ", chaine[-1])
```

```
Dernière lettre :  e
```

In [11]:

```
print("Les deux premières lettres :", chaine[:2])
```

```
Les deux premières lettres : ma
```

In [12]:

```
print("Les six dernières lettres :", chaine[-6:])
```

```
Les six dernières lettres : chaîne
```

In [13]:

```
print("Une partie de la chaîne :", chaine[3:8])
```

```
Une partie de la chaîne : belle
```

4. Quelques méthodes (https://docs.python.org/fr/3.7/library/stdtypes.html#text-sequence-type-str) de chaînes de caractères

Une méthode est une fonction qui est appelée sur les `instances` : les objets d'un type donné.

In [14]:

```
'voici une phrase'.count('i')  # Pour compter des occurences
```

Out[14]:

```
2
```

La méthode `count()` s'applique à l'objet `'voici une phrase'` qui est de type `str`.
`'i'` est une argument de cette méthode.

In [15]:

```
n = 7
'la variable n vaut {0} et la suivante {1}, puis un mot : {2}'.format(n, n+2, "BIPBIP")
```

Out[15]:

```
'la variable n vaut 7 et la suivante 9, puis un mot : BIPBIP'
```

In [16]:

```
'pour écrire en majuscules'.upper()
```

Out[16]:

```
'POUR ÉCRIRE EN MAJUSCULES'
```

In [17]:

```
'EN mINuscules'.lower()
```

Out[17]:

```
'en minuscules'
```

On peut également savoir si une chaîne est présente dans la chaîne de caractères.

In [18]:

```
'e' in 'mot'
```

Out[18]:

```
False
```

In [19]:

```
'er' in 'calculer'
```

Out[19]:

```
True
```

5. Quelques fonctions s'appliquant à ces objets

In [20]:

```
type('mot')
```

Out[20]:

```
str
```

In [21]:

```
len('mot')
```

Out[21]:

```
3
```

In [22]:

```
min('variable')
```

Out[22]:

```
'a'
```

In [23]:

```
max('variable')
```

Out[23]:

```
'v'
```

In [24]:

```
int('5') # Transtypage en nombre entier
```

Out[24]:

```
5
```

In [25]:

```
float('4') # Transtypage vers un nombre décimal
```

Out[25]:

```
4.0
```

III - Types séquentiels : list (https://docs.python.org/fr/3.7/library/stdtypes.html#lists), tuple (https://docs.python.org/fr/3.7/library/stdtypes.html#tuples)

1. Les séquences muables de type `list`

Python possède des types composés qui sont utilisés pour regrouper ensemble des variables de différents types. Le plus polyvalent sont les listes (de type `list`), qui peuvent être écrits comme une suite de valeurs séparées par des virgules le tout encadré par des crochets.

In [26]:

```
l = [] # La liste vide
```

In [27]:

```
len(l)
```

Out[27]:

```
0
```

In [28]:

```
l2 = [3, 'e', 3.9]  # Une liste de trois éléments différents
```

Comme pour les chaînes de caractères, on peut accéder aux éléments de d'une liste : le premier élément est celui d'indice 0 :

In [29]:

```
l2[0]
```

Out[29]:

```
3
```

In [30]:

```
l2[-1] # Le dernier
```

Out[30]:

```
3.9
```

On peut **modifier** (on dit qu'une liste est `muable`) un élément de la liste à l'aide d'une affectation :

In [31]:

```
l2[1] = 7
print(l2)
```

```
[3, 7, 3.9]
```

In [32]:

```
[1,2]+[3,4] # Pour concaténer deux listes
```

Out[32]:

```
[1, 2, 3, 4]
```

Quelques méthodes importantes :

In [33]:

```
l2.append(1)
print(l2)
```

```
[3, 7, 3.9, 1]
```

In [34]:

```
p = l2.pop()
print(p, " et ", l2)
```

```
1  et  [3, 7, 3.9]
```

In [35]:

```
l2.reverse()
print(p, " et ", l2)
```

```
1  et  [3.9, 7, 3]
```

Liste de listes

Les éléments d'une liste peuvent être des listes. Cela donne des tableaux à deux dimensions. Il peuvent être très utiles pour représenter des images, des plateaux de jeux, et bien d'autres choses encore.

Voici ci dessous un exemple ou `a` et `n` sont deux listes, et la liste `x` les contient. Regardez comment sont utilisés les indices.

In [36]:

```
a = ['a', 'b', 'c']
n = [1, 2, 3]
x = [a, n]
x
```

Out[36]:

```
[['a', 'b', 'c'], [1, 2, 3]]
```

In [37]:

```
x[0]
```

Out[37]:

```
['a', 'b', 'c']
```

In [38]:

```
x[0][1]
```

Out[38]:

```
'b'
```

In [39]:

```
x[1][0]
```

Out[39]:

```
1
```

2. Les séquences immuables de type `tuple` (https://docs.python.org/fr/3.7/library/stdtypes.html#tuples)

Les tuples (n-uplets en français) sont des séquences immuables, généralement utilisées pour stocker des collections de données hétérogènes. On le définie par une suite de valeurs séparées par des virgules le tout encadré par des parenthèses.

In [40]:

```
t = () # Tuple vide
```

In [41]:

```
type(t)
```

Out[41]:

```
tuple
```

In [42]:

```
len(t)
```

Out[42]:

```
0
```

In [43]:

```
t1 = ("3", [1, 2], 2e7)
```

In [44]:

```
t1[1]
```

Out[44]:

```
[1, 2]
```

In [45]:

```
t1[1][1]
```

Out[45]:

```
2
```

In [46]:

```
t1[0] = 1 # Un tuple est immuable, cette instruction renvoie une erreur
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-46-c0c4497a27ce> in <module>
----> 1 t1[0] = 1 # Un tuple est immuable, cette instruction renvoie une e
rreur

TypeError: 'tuple' object does not support item assignment
```

IV - Récapitulatif des opérations sur les séquences muables

Opération	Résultat
`s[i] = x`	élément `i` de `s` est remplacé par `x`
`s[i:j] = t`	tranche de `s` de `i` à `j` est remplacée par le contenu de l'itérable `t`
`del s[i:j]`	identique à `s[i:j] = []`
`s.append(x)`	ajoute `x` à la fin de la séquence (identique à `s[len(s):len(s)] = [x]`)
`s.copy()`	crée une copie superficielle de `s` (identique à `s[:]`)
`s.extend(t)` ou `s = s + t`	étend `s` avec le contenu de `t`
`s.insert(i, x)`	insère `x` dans `s` à l'index donné par `i`
`s.pop([i])`	récupère l'élément à `i` et le supprime de `s`
`s.remove(x)`	supprime le premier élément de `s` pour lequel `s[i]` est égal à `x`
`s.reverse()`	inverse sur place les éléments de `s`