## Types de base

### Les entiers :int

```
42   -1112   0b110   0xF3
```

0b110 : binaire (6) ; 0xF3 : hexa (243)

### Les flottants : float

```
11.09   -11.1019e81
```

$-11.1019 \times 10^{81}$

**Opérations avec `int` et `float`**

| `+` | `-` | `*` | `/` | `//` | `%` | `**` |
|---|---|---|---|---|---|---|
| + | − | × | ÷ | quotient | reste | puissance |

```
>>> (2+4*2)**2
100
>>> 16/3
5.333333333333333
>>> print(16//3 , 16%3)
5 1
>>> abs(-3.14)
3.14
>>> round(21.091981,2)
21.09
```

### Les booléens : bool

```
True   False
```

### Les chaînes de caractères : str (strings)

```
'Hello World' "L'apostrophe"   ""
```

"" : Une liste vide

```
>>> print("Ligne1\nLigne2\tet une tabulation")
Ligne1
Ligne2    et une tabulation
```

(`\n` retour chariot et `\t` pour la tabulation)

### Immuables

Un type est immuable si la valeur d'une variable de ce type ne peut changer que par l'affectation d'une nouvelle valeur à cette variable. Dans le cas contraire, il sera muable.

## Variables

### Affectation

```
a = 42
b = 12 + 3*4
text1=text2='coucou'
a,b,c=2,3.14,"hello"
```

`b = 12 + 3*4` : evaluation de l'expression puis affectation

`text1=text2='coucou'` : Affectation même valeur

`a,b,c=2,3.14,"hello"` : Affectations multiples (voir tuples)

### Noms de variables

Noms de variables 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.
Un nom de variable ne doit pas débuter par un chiffre et il n'est pas recommandé de le faire débuter par le caractère _ (sauf cas très particuliers) .Python est sensible à la casse, les variables test, Test et TesT sont toutes différentes. Ne pas utiliser les mots "réservés" comme print, for...

## Affectation et opérations

```
>>> mon_nombre = 40
>>> mon_nombre += 2
>>> #equivaut à mon_nombre = mon_nombre + 2
>>> print(mon_nombre)
42
>>> mon_nombre -= 10
>>> print(mon_nombre)
32
>>> # Autres possibilités : *= /= **= //= %=
```

## conversions

| | | |
|---|---|---|
| `type('152')` | `<class 'str'>` | Voir le type |
| `int('152')` | `152 <class 'int'>` | Conversion vers int |
| `int('11011',2)` | 27 | Second param = base |
| `float('3.1415e6')` | 3141500.0 | Conversion vers float |
| `str(21)` | 21 <class 'str'> | Conversion vers string, voir **formatage** |
| `chr(65)` | A | conversion code ASCII vers caractère |
| `ord('B')` | 66 | conversion caractère vers code ASCII |
| `bool(x)` | `False` pour `x` zéro ou vide ou `None`, `True` pour autres | |
| `list('coucou')` | ['c', 'o', 'u', 'c', 'o', 'u'] | conversion en list (fonctionne avec les tuples, dict et set) |
| `set('coucou')` | 'o', 'c', 'u' | conversion en ensemble (fonctionne avec les tuples, dict et list) |
| `tuple('coucou')` | ('c', 'o', 'u', 'c', 'o', 'u') | conversion en tuple (fonctionne avec les set, dict et list) |
| `'-'.join(['06','01','02'])` | 06-01-02 | Jointure de séquences de `str` |
| `'le la les'.split()` | ['le', 'la', 'les'] | découpage de str entre espaces |
| `'06-01-02'.split('-')` | ['06', '01', '02'] | découpage de str, le séparateur du découpage peut être choisi en paramètre |

## Types construits

- séquences ordonnées : accès par l'index
  - `list` **muables**

    `[42,24,5]` `['Un','Deux']` `[True]` `une_liste = []` (Initialisation liste vide)
  - `tuple` **immuables**

    `(42,24,5)` `a = 'Un','Deux'` (fonctionne sans ()) `un_tuple = 1,2` (Attention 1,2≠1.2)
- Conteneur, clés : accès rapide par clés , chaque clé doit être unique et immutable
  - `dict` **- dictionnaire - muable**

    `{'cle':'valeur'}` `{'nom':'Ho','taille':1.75,'ami':True}`

```
>>> a = dict(one=1, two=2, three=3)
>>> print(a)
{'one': 1, 'two': 2, 'three': 3}
```

    `dico = {}` (Initialisation dict vide)
- Ensembles, valeurs uniques (élimine les doublons rapidement)

  `set` **- ensembles - muable**

  `frozenset` **- ensembles gelés - immuable**

  `{'val1','val2','val3'}` `set([42,73,105,12])` `frozenset(['a','b','c','d'])`

```
>>> s = {1,1,2,3,2,4,1}
>>> print(s)
{1, 2, 3, 4}
```

## Indexation et slice

### Indices

Accès aux éléments de `list`, `str` , `tuple` via un indice. Les indices démarrent à **zéro** .

| | | | | |
|---|---|---|---|---|
| liste : | ['girafe', 'tigre', 'singe', 'souris'] | | | |
| indice positif : | 0 | 1 | 2 | 3 |
| indice négatif : | −4 | −3 | −2 | −1 |

Exemples :

```
>>> liste= ['girafe', 'tigre', 'singe', 'souris']
>>> print(liste[0],liste[1],liste[3])
girafe tigre souris
>>> print(liste[-1],liste[-2],liste[-4])
souris singe girafe
>>> #la fonction len() permet de récupérer la taille d'une liste
>>> print(len(liste),liste[len(liste)-1])
4 souris
```

### slicing ou tranches

Utiliser `liste[tranche_debut:tranche_fin:pas]` permet d'accéder à des sous-listes

```
>>> sac = ['clés','portable','tee-shirt','eau','chaussures']
>>> sac[1:3]
['portable', 'tee-shirt']
>>> sac[1:]
['portable', 'tee-shirt', 'eau', 'chaussures']
>>> sac[:-2]
['clés', 'portable', 'tee-shirt']
>>> sac[1::2]
['portable', 'eau']
>>> sac[::-1]
['chaussures', 'eau', 'tee-shirt', 'portable', 'clés']
>>> # Copie superficielle (shallow copy)
>>> sac2 = sac[:]
```

## Opération sur les booléens

### ET, OU et NON logiques

```
>>> print(True and False , True or False , not True)
False True False
>>> # Évaluation au plus court (ou fainéante), 'un truc'+faux ne sera pas
>>> # évalué car True OU ? donnera True
>>> print(True or 'un truc'+faux)
True
```

### Comparaison

| | | | | | |
|---|---|---|---|---|---|
| `<` | `>` | `<=` | `>=` | `==` | `!=` |
| < | > | ⩽ | ⩾ | = | ≠ |

```
>>> affirmation1 = (1 < 3) # parenthèses pour la lisibilité
>>> affirmation2 = (2 >= 15)
>>> affirmation1 == affirmation2
False
>>> print(affirmation1,affirmation2)
True False
```

## Instruction conditionnelle

L'instruction conditionnelle `if` permet de tester des conditions et d'éxécuter des instructions en fonction des résultats de ces tests. (Merci Ada Lovelace !)

### if

```
if age >= 18: # Le bloc indenté n'est exécuté que si la condition est vraie
    print('Vote possible')
print('Ici, hors du bloc conditionnel')
```

Variante, un booléen peut être la condition

```
condition = (age <18)
if not condition:
    print('Vote possible')
```

### if-else

```
if age < 16:
    print('Pas  de conduite')
elif 16 <= age < 18: # (16 <= age) and (age <18)
    print('Conduite  accompagnée')
else:
    print('permis de conduire')
```

## Boucles non bornées

### While

Lorsque l'on veut répéter une suite d'instructions sans savoir combien de tours de boucle il faut effectuer alors il faudra utiliser une boucle non bornée. C'est la boucle while (tant que en Français !).
L'instruction `while` permet de tester une condition et d'exécuter les instructions indentées tant que la condition est vraie

```
>>> compteur = 10
>>> while compteur > 0:
...     print(compteur, end=' ')
...     compteur -= 1
...
10 9 8 7 6 5 4 3 2 1
```

## Boucles bornées

### Boucles bornées `for`

Parcours séquentiel d'une liste

**Par parcours des éléments de la liste**

```
>>> liste = ['a','b','c']
>>> for elem in liste:
...     print(elem, end=' ')
...
a b c >>> # end=' ' Voir Input/output
```

**Par parcours des index des éléments de la liste**

```
>>> liste = ['a','b','c']
>>> for indice in range(len(liste)):
...     print(liste[indice], end=' ')
...
a b c
```

### range : séquences d'entiers

`range([debut,]fin[,pas])` : début 0 par défaut, fin non compris dans la séquence et pas par défaut 1. [...] : paramètre facultafif, si vide prendra la valeur par défaut.

```
>>> for k in range(6):
...     print(k,end = ' ')
...
0 1 2 3 4 5
```

```
>>> for k in range(2,8):
...     print(k,end = ' ')
...
2 3 4 5 6 7
```

```
>>> for k in range(-2,8,2):
...     print(k,end = ' ')
...
-2 0 2 4 6
```

```
>>> for k in range(20,4,-5):
...     print(k,end = ' ')
...
20 15 10 5
```

`range` ne fournit pas une liste, c'est un itérable, mais il peut être converti en liste (voir **conversions**)

```
>>> list(range(11))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

## Input/Output

### Affichage : print

`print(elements_a_afficher[,sep][,end])` : affiche les elements_a_afficher, converti en str si nécessaire (voir conversion).
sep :séparateur des éléments, par défaut : espace.
end : fin d'affichage, par défaut : retour à la ligne.

```
>>> nom,prenom,taille = 'VADOR','Dark',2.03
>>> print('Je suis',prenom,nom,'et je mesure',taille,' m')
Je suis Dark VADOR et je mesure 2.03  m
>>> for k in range(5,0,-1):
...     print(k,end = '-')
...
5-4-3-2-1-
>>> for k in range(5, 0, -1):
...     print(k, k**2, end=' - ', sep=':')
...
5:25 - 4:16 - 3:9 - 2:4 - 1:1 -
```

### Saisie utilisateur : input

```
reponse = input('Entrez votre nom :')
age = input('Donnez votre age : ')
```

`input` retourne un objet de type `str`. Dans l'exemple sur l'age, pour calculer avec la variable `age` il faudra d'abord la convertir en `int` ou `float` (voir **conversions**)

## Fonctions

### Fonction

```
def ma_fonction(mes_parametres):
    ''' documentation '''  # sera affiché lors de l'appel de help(ma_fonction)
    # instructions executées à l'appel de la fonction
    return resultat # le retour peut être de tout type : booléen, str, int ,list ...
```

Si pas de valeur à retourner : `return None` Les variables et paramètres effectifs dans le bloc fonction sont définis pendant l'exécution de la fonction. Une variable définie dans le corps d'une fonction sera locale à celle-ci et ne pourra être accessible hors de la fonction sauf si elle est définie avec le mot clé `global`, mais cet usage est à risque. (Voir effet de bord)

**Exemples**

```
>>> def salutation1(nom):
...     """
...     Renvoie une salutation
...     paramètres :
...     nom : str
...     """
...     return "Bonjour "+nom
...
>>> #appel de la fonction
>>> hello = salutation1("Luke")
>>> #Ici, le retour de la
>>> # fonction est affecté
>>> # à la variable hello
>>> print(hello)
Bonjour Luke
```

```
>>> def salutation2(nom,lien='père'):
...     """
...     Renvoie une salutation
...     paramètres :
...     nom : str
...     lien : str, default = 'père'
...     """
...     return "Bonjour "+nom+", je suis ton "+lien
...
>>> #Second paramètre vide donc valeur par défaut
>>> print(salutation2("Luke"))
Bonjour Luke, je suis ton père
>>> #Second paramètre non vide donc utilisé
>>> print(salutation2("Luke",'ami'))
Bonjour Luke, je suis ton ami
```

```
>>> def min_max(liste):
...     """
...     Retourne les plus petits
...     et plus grands élements
...     de la liste
...     paramètre : une liste non
...     vide de flottants
...     """
...     # initialisation
...     mini, maxi = liste[0], liste[0]
...     for elem in liste:
...         if elem < mini:
...             mini = elem
...         elif elem > maxi:
...             maxi = elem
...     # Le retour sera un tuple
...     return mini, maxi
...
>>> min_max([-2, 321, 15, -5.2])
(-5.2, 321)
```

### Test : assert

Des tests bien construits aideront à éviter certaines erreurs et à maintenir correctement un code complexe. Plusieurs méthodes sont possible, une très simple est basée sur `assert`

`assert booléen` : si booléen est `True` le code se déroulera sans erreur et sinon une erreur sera affichée (`AssertionError`).

```
>>> assert salutation1('BOB') == 'Bonjour BOB'
>>> assert salutation1('Luke') == 'Bonjour Yoda'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

## Opération sur les chaînes de caractères - `str`

### Méthodes

```
>>> # Concaténation avec l'opérateur $+$
>>> 'Coucou' + ' le monde'
'Coucou le monde'
>>> chaine = "Une chaîne de caractères"
>>> chaine.upper()
'UNE CHAÎNE DE CARACTÈRES'
>>> chaine.lower()
'une chaîne de caractères'
>>> 'dupond'.capitalize()
'Dupond'
>>> chaine.split()
['Une', 'chaîne', 'de', 'caractères']
>>> chaine.count('a')
3
>>> chaine.find('cha')
4
>>> chaine.replace('ne','me')
'Ume chaîme de caractères'
```

Lien vers toutes les méthodes de la classe str

### Formatage

```
>>> 'La somme de 1 et 2 est : {0}'.format(1+2)
'La somme de 1 et 2 est : 3'
>>> # Chaque {.} est remplacée par le paramètre d'indice correspondant
>>> 'Mon nom est {0}, {0} {1}'.format('James','BOND')
'Mon nom est James, James BOND'
>>> cote = 3
>>> texte = "L'aire du carré de côté {0} est : {1} cm²"
>>> texte.format(cote,cote**2)
"L'aire du carré de côté 3 est : 9 cm²"
>>> '{prod} : {prix} €'.format(prix=1.15,prod='baguette')
'baguette : 1.15 €'
>>> # Possibilités de spécifier un format spécifique
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
```

Compléments : ici

## Opération sur les listes - `list`

### Méthodes

```
>>> liste = ['ari','bob','cara','dan']
>>> # Remplacer un élément
>>> liste[0] = 'al'
>>> print(liste)
['al', 'bob', 'cara', 'dan']
>>> # Ajouter un élement en fin de liste
>>> liste.append('ed')
>>> print(liste)
['al', 'bob', 'cara', 'dan', 'ed']
>>> # Insertion à un index donné
>>> liste.insert(2,'ben')
>>> print(liste)
['al', 'bob', 'ben', 'cara', 'dan', 'ed']
>>> liste.remove('ben')
>>> print(liste)
['al', 'bob', 'cara', 'dan', 'ed']
>>> # Supprime l'élément d'index donné
>>> # par défaut supprime le dernier
>>> liste.pop()
'ed'
>>> print(liste)
['al', 'bob', 'cara', 'dan']
>>> liste.pop(1)
'bob'
>>> print(liste)
['al', 'cara', 'dan']
>>> ## Tri pas en place
>>> # (une nouvelle liste triée est crée)
>>> desord = ['b','c','a']
>>> nouv_liste = sorted(desord)
>>> print(desord,nouv_liste)
['b', 'c', 'a'] ['a', 'b', 'c']
>>> # Tri en place
>>> # (la liste initiale est trié)
>>> desord.sort()
>>> print(desord)
['a', 'b', 'c']
```

Lien vers toutes les méthodes de la classe list

### Listes en compréhension

Objectif : générer des listes de manières concises et lisibles. Il est toujours possible de formater ces codes avec des boucles et conditionnelles.

**Syntaxe**

```
[fonction(elem) for elem in list if condition]
```

**Exemples**

```
>>> carres = [x**2 for x in range(1,11)]
>>> print(carres)
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> cubes_pairs = [x**3 for x in range(1,21) if x%2 == 0]
>>> print(cubes_pairs)
[8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832, 8000]
>>> list1 = [1,4,2,7,1,9,0,3,4,6,6,6,8,3]
>>> list2 = [elem for elem in list1 if elem>5]
>>> print(list2)
[7, 9, 6, 6, 6, 8]
```

## Opération sur les dictionnaires - `dict`

### Méthodes

```
>>> placard = {'bol':2,'verre':5,'tasse':4}
>>> # Accès par clé
>>> placard['verre']
5
>>> # Taille dictionnaire
>>> len(placard)
3
>>> #Remplacement valeur
>>> placard['tasse']=8
>>> print(placard)
{'bol': 2, 'verre': 5, 'tasse': 8}
>>> # Supprimer un élément (voir list)
>>> placard.pop('bol')
2
>>> print(placard)
{'verre': 5, 'tasse': 8}
>>> d = {1: "one", 2: "three"}
>>> d1 = {2: "two",3:'three'}
>>> d.update(d1)
>>> print(d)
{1: 'one', 2: 'two', 3: 'three'}
```

Lien vers toutes les méthodes de la classe dict

### Boucles sur dictionnaires

```
>>> placard = {'bol':2,'verre':5,'tasse':4}
>>> # Parcours par clés
>>> for cle in placard:
...     print(cle,placard[cle])
...
bol 2
verre 5
tasse 4
>>> # Parcours par valeurs
>>> for valeur in placard.values():
...     print(valeur)
...
2
5
4
>>> # Parcours par paire clé/valeurs
>>> for cle,valeur in placard.items():
...     print(cle,valeur)
...
bol 2
verre 5
tasse 4
```

## Opération sur les ensembles - `set`

### Méthodes

```
>>> s1 = {2,4,6}
>>> s2 = {3,4,5,7}
>>> # Test appartenance
>>> print(2 in s1 , 2 in s2)
True False
>>> # Les opérations mathématiques
>>> # sur les ensembles sont implémentées
>>> # union
>>> print(s1 | s2)
{2, 3, 4, 5, 6, 7}
>>> # intersection
>>> print(s1 & s2)
{4}
>>> # différence
>>> print(s2-s1)
{3, 5, 7}
>>> # union privé de l'intersection
>>> print(s2 ^ s1)
{2, 3, 5, 6, 7}
>>> # ajout d'élément
>>> s1.add(8)
>>> print(s1)
{8, 2, 4, 6}
>>> # Suppression (erreur si elem abs)
>>> s1.remove(6)
>>> print(s1)
{8, 2, 4}
>>> # Suppression si elem présent
>>> s1.discard(91)
>>> s1.remove(91)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 91
```

Lien vers toutes les méthodes de la classe set

## Modules

### Import de modules

```
>>> # import de tout un module
>>> import random
>>> # Accès au méthodes via la syntaxe suivante :
>>> random.randint(1,6) # Simuler un entier aléatoire
6

>>> # import de certaines méthodes d'un module
>>> from random import choice
>>> # Appel de la methode sans utiliser le nom du module
>>> print(choice(['a','b','c'])) # tirage aléatoire d'un élément dans une séquence
c

>>> # import d'un module perso dans le même dossier (mon_module.py)
>>> # import mon_module
```

Lien vers la doc du module random

### module maths

```
>>> from math import pi,cos,sqrt,log,e
>>> cos(pi)
-1.0
>>> sqrt(49)
7.0
>>> log(e**3)
3.0
>>> from math import ceil,floor
>>> ceil(10.5)
11
>>> floor(10.5)
10
```

Lien vers la doc du module maths