# Initiation Python

## 19 au 22 septembre 2016

*Léopold GAMBA*

*contact@formaself.com*

*Paris*

# Pré-requis

1. **Savoir utiliser un PC sous Windows**
2. **Premières notions de programmation (un plus)**

Objectif : acquérir les notions de base pour programmer


1- Introduction & historique

2- La syntaxe du langage

3- POO en Python

4- Autres sujets

## 1 - Historique , caractéristiques

a- Langage créé en 1990 par Guido Von Rossum

b- Implémentation libre et open source.

c- Dernière version Python 3.5 (implémentation CPython).

Multi paradigme
- Code fonctionnel
- Programmation impérative
- Programmation OO

Langage de haut niveau
- Gestion de la mémoire
- Typage des variables

*( abstraction des détails de bas niveau )*

## *2-* Spécificités du langage

Le langage Python est :

a- Lisible (proche du langage naturel)

b- Simple à apprendre

c- Concis

*Code en PHP*

```php
//BOUCLE WHILE
$i = 0;
while(i < 5){
    print i;
    $i++;
}
print i;
```

*Code en Python*

```python
# BOUCLE WHILE
i = 0
while i < 5:
    print(i)
    i += 1
print(i)
```

## *3-* Forces du langage Python

a- Vitesse de développement et maintenance

b- Communauté forte et active

c- Multi-plateforme (Windows, Linux, Mac, etc.)

## *4-* Faiblesses du langage Python

a- Performances d'exécution

b- Distribution

## *5-* Versions du langage Python

Deux versions majeures :

a- Version 2.x (*à abandonner en 2020*)

b- Version 3.x

***Nota***

1. Les deux versions sont incompatibles.
2. Deux versions peuvent cohabiter sur une machine.

## *6-* Qui utilise Python & que faire avec?

De grandes entreprises + grands secteurs de l'industrie + etc :

1. Langage pour apprendre la programmation en prépas
2. Youtube a été recodé en Python
3. DropBox a embauché le créateur de Python
4. La NASA l'a utilisé pour faire des simulations
5. Instagram a été programmé en Django
6. Administration système
7. Dans les Banques
8. Scientifiques
9. Géographie, etc.

On peut écrire :

- Scripts systèmes, admin serveurs
- Sites web : Django, Pyramid, etc.
- Programmes scientifiques, calcul
- Etc.

## 7- Un programme minimal

```
print("Hello, World!")
```

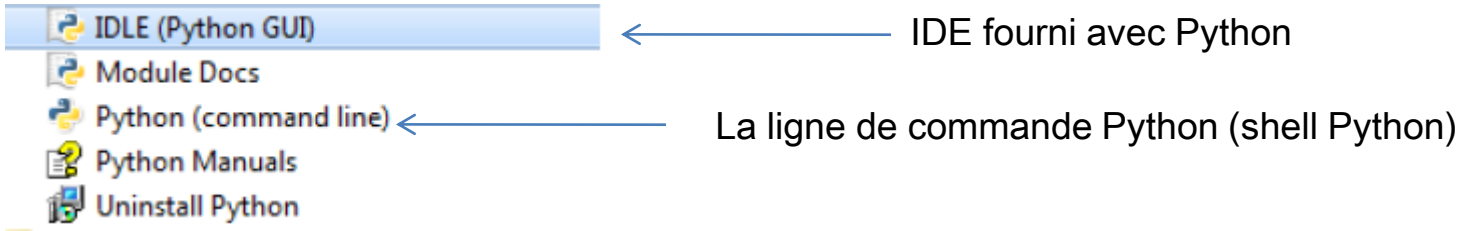
## *1-* Télécharger et installer Python

### Travaux pratiques

a- Télécharger ici la version 3.5 Windows 64 bits : www.python.org

b- Installer en suivant les étapes
*Nota* : ajouter Python 3.5 au PATH

c- Notez le dossier d'installation




*Menu Démarrer > Python*

## *2-* Différents éditeurs et IDE et shell

### Editeurs

| | |
|---|---|
| a- PyCharm | pycharm.org |
| b- Sublime Text | sublime.com |
| c- Notepad++ | notepadplus.org |
| d- Eclipse associé à PyDev | eclipse.com |
| e- IDLE | inclus dans Python |

etc.

### Shells

Linux (bash, etc.)
Cmd sous Windows
Console2 + module pyreadline + IPython

## *3-* Lancer un code Python

### Exercice 1 : taper du code Python

a- Lancer le terminal Windows avec cmd puis c:\>python à l'invite.

b- Le shell Python **>>>** s'affiche. Tapez print("Hello, World")

c- Puis validez.

### Exercice 2 : lancer un script Python

a- Créer c:\>stage\test.py avec l'éditeur Notepad++ ou autre, avec print("Hello, World") dedans.

b- Lancer cmd depuis c:\>stage puis python test.py

## *4-* Terminal et shell améliorés

### Installer Console2 + pyreadmine + IPython

a- Télécharger Console2 et dézippez-le dans Program Files

b- Lancez Console2 puis à l'invite : c:\>pip install pyreadline

c- Installer IPython : c:\>pip install ipython

d- Lancer ipython dans Console2 : c:\>ipython

e- Le shell amélioré s'affiche.

## *5-* Créer un environnement virtuel

### Installer Console2 + pyreadmine + IPython

a- Dans Console2 : c:\>pip install virtualenv

b- Puis : c:\>pip install virtualenvwrapper-win (pour Windows)

c- Installer l'environnement : c:\>mkvirtualenv --python=chemin/vers/dossier/python stage
Le chemin vous permet de préciser quel version de l'interpréteur choisir.

d- Pour supprimer l'environnement : c:\> rmvvirtualenv stage

e- Sélectionner l'environnement virtuel pour y travailler : c:\> workon stage

f- deactivate permet de sortir d'un environnement virtuel.

g- Lister les modules installés dans l'environnement virtuel et en faire un prérequis :
(stage) pip3 freeze > requirement.txt

## 6- Installer des modules avec PIP

PIP est le logiciel chargé de gérer les modules Python (paquets) de votre système. Une large partie des modules Python est hébergée sur PyPi (Python Package index).
PIP est installé avec Python.

### Commandes PIP

install : installer un module et l'ensemble de ses dépendances. *Exemple : pip install requests*
uninstall : désinstalle un module. *Exemple : pip uninstall requests*

-r : avec cette option, on installe un ensemble de modules contenus dans un fichier.
*Exemple : pip install –r requirements.txt*

-u : avec cette option, on met à jour un module déjà installé.
*Exemple : pip install –u requests*

list : liste les modules installés sur le système.
*Exemple : pip3 list*

search : recherche dans la description des modules hébergés par PyPi.
*Exemple : pip3 search GUI*

## 7- Télécharger et installer un éditeur de code

### Travaux pratiques

a- Télécharger Notepad++ ou Sublime

b- Installer en suivant les étapes

c- Créer c:\>stage\test_hello.py avec print("Hello, World") dedans.

d- Tester le fichier

## *1-* Shell Python : affichage, arithmétique, vocabulaire (*fonction, variable, mots clés, objets*)

### Travaux pratiques

a- Lancer le shell Python (Console2 + iPython).

b- exécuter les instructions ci-dessous, et observez.

| Instruction | Sortie | Commentaire |
|---|---|---|
| `print("Hello, World!")` | | |
| `print('Hello, World!')` | | |
| 1+3 | | |
| 3-1 | | |
| -6 | | |
| 2*4 | | |

## *1-* Shell Python : affichage, arithmétique, vocabulaire : suite (*fonction, variable, mots clés, objets*)

| Instruction | Sortie | Commentaire |
|---|---|---|
| 2**3 | | |
| 3/2 | | |
| 3//2 | | |
| 3.0 | | |
| nombre = 125 | | |
| nombre | | |
| mot = "Coucou" | | |
| mot | | |
| del mot | | |
| mot | | |

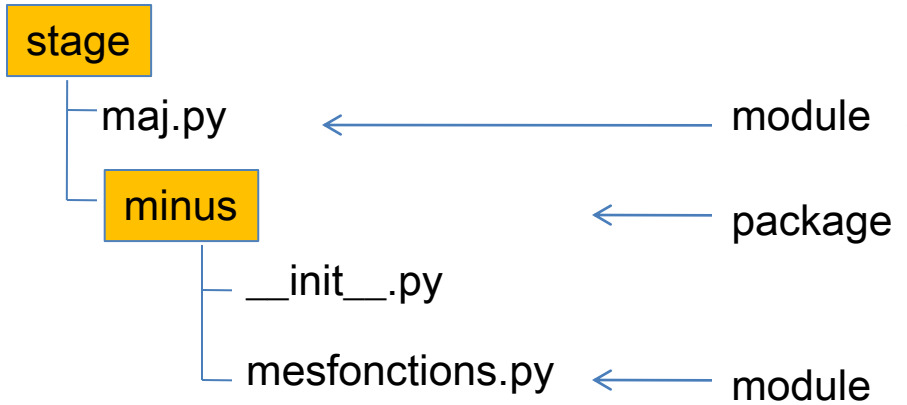
## *1-* Shell Python : affichage, arithmétique, vocabulaire : suite 1 (*fonction, variable, mots clés, objets*)

| Instruction | Sortie | Commentaire |
|---|---|---|
| mot = "Cool" | | |
| mot.upper() | | |
| mot.lower() | | |
| x = 3 | | |
| print(x) | | |
| a,b,c = 3,5,7 | | déclaration de 3 variables a, b et c de valeurs resp. 3, 5 et 7 |
| print(a+b) | | |
| print('la valeur de', a, '+', b, 'est :', a+b) | | |
| a-b/c | | |
| (a-b)/c | | |

## *1-* Shell Python : affichage, arithmétique, vocabulaire : suite 2 (*fonction, variable, mots clés, objets*)

| Instruction | Sortie | Commentaire |
|---|---|---|
| b%c | | modulo |
| max([4, 8, 9, 14, 12]) | | |
| m = max([4, 8, 9, 14, 12]) | | Le plus grand des nombres d'un tableau (liste) |
| m | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 1- Utiliser les modules importés




```
# Importation
from maj import maj
from minus import mesfonctions

# Utiliser le 1er module
print(maj("Coucou"))

# Utiliser le 2ème module
print(mesfonctions.minuscule("COUCOU"))
```

*Importer les modules et les utiliser*

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#
# maj.py
#
""" Un super module """

# Une fonction
def maj(chaine):
    return chaine.upper()
```

*maj.py*

```
# Une fonction
def minuscule(chaine):
    return chaine.lower()
```

*mesfonctions.py*

## 2- Utiliser un paquet : module ou ensemble de modules

### Installer le paquet distant requests

a- Installer avec PIP : C:\> pip install requests

b- utilisation

```
# Importer requests
import requests

# Utiliser
print(requests.get('http://www.formaself.com).headers)
```

### Utiliser le paquet math

```
# Importer module existant
from math import sqrt

# Utiliser
print(sqrt(4))
```

## 3- Quelques modules courants

– math : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
– sys : fournit les paramètres et fonctions liées à l'environnement d'exécution : passage d'arguments, gestion de l'entrée/sortie standard...
– os : dialogue avec le système d'exploitation (e.g. permet de sortir de Python, lancer une commande en shell, puis de revenir à Python).

– random : génération de nombres aléatoires.
– time : permet d'accéder à l'heure de l'ordinateur et aux fonctions gérant le temps.
– calendar : fonctions de calendrier.
– profile : évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).

– urllib2 : permet de récupérer des données sur internet depuis python.
– Tkinter : interface python avec Tk (permet de créer des objets graphiques ; nécessite d'installer Tk).
– re : gestion des expressions régulières.
– pickle : écriture et lecture de structures Python (comme les dictionnaires par exemple).

– string : fournit des opérations courantes sur les chaînes de caractères (équivalentes aux méthodes de la classe string plus quelques bonus) : *string.lower('FOO')*, puis *upper()*, *capitalize()*, *capwords()*, *strip()*, *expandtabs()*. Il est généralement recommandé d'utiliser les méthodes disponibles sur les objets de type string équivalentes.

## 1- Découvrir les différents types : *fonction : type(), isinstance(.)*

### Travaux pratiques

| Instruction | Sortie | Commentaire |
|---|---|---|
| a = 12 | | |
| type(a) | | |
| b = 3.1 | | |
| type(b) | | |
| isinstance(a,int) | | Un entier |
| isinstance(b,int) | | |
| isinstance(b,float) | | Un float |
| isinstance("coucou" , str) | | Une chaîne |
| type(True) | | Un booléen |
| isinstance(True , bool) | | |

## 2- Variables et types de retour, retour de print()

De quel type est le retour d'une fonction ?

Travaux pratiques

| Instruction | Sortie | Commentaire |
|---|---|---|
| a = max([2, 5, 12, 1, 3]) | | |
| a | 12 | |
| type(a) | <class 'int'> | La fonction max a retourné un entier. |
| mot = print("Cool ") | | |
| type(mot) | <class 'NoneType'> | print ne retourne rien (type NoneType) |
| print | <built-in function print> | |
| | | |
| | | |

## 1- Les nombres entiers

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| int() | | Le constructeur |
| chiffre = -51 | | |
| chiffre | | |
| isinstance(chiffre, int) | | |
| isinstance("Hello", int) | | |
| 2 + 2 | | L'addition |
| 3/2 | | Un flottant retourné. |
| 4/2 | 2.0 | La division retourne tjs 1 nb à virgule (flottant) |
| isinstance(4/2, int) | False | |
| isinstance(4/2, float) | True | |

## 1- Les nombres entiers (suite)

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| 2*2 | | Multiplication |
| 2**3 | 8 | Puissance. Retourne un entier. |
| chiffre | | |
| myint = int(15) | | Conversion en int avec int() |
| myint | 15 | |
| | | |
| | | |
| | | |
| | | |

## 2- Les nombres flottants

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| 2.0 | 2.0 | |
| isinstance(2.0, float) | True | |
| float(2) | 2.0 | Conversion d'un entier en flottant |
| int(1.5) | 1 | Arrondi à l'entier le plus petit |
| int(-1.5) | -1 | |
| int(-3/2) | -1 | |
| 3//2 | 1 | Division entière (retourne un entier) |
| -3//2 | -2 | Si troncature : retourne le plus petit nombre |

## 2- Les nombres flottants (suite)

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| print(0.1) | 0.1 | Illusion donnée par Python. |
| print(0.3 – 0.2) | 0.9999 | |
| Utiliser le module **decimal** | | |
| from decimal import decimal as deci | 1 | Arrondi à l'entier le plus petit |
| deci(0.1) | Decimal('0.00') | Attention aux performances |
| deci("0.1") | | Représente 0.1 de façon précise |
| | | |
| | | |

## 1- Le type chaine de caractères

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| chaine = "Chaîne de Kres" | | Doubles ou simples quotes, c'est pareil |
| chaine | 'Chaîne de Kres' | |
| type(chaine) | <class 'str'> | |
| isinstance(chaine, str) | True | |
| "Hello" + 'World' | 'HelloWorld' | Concaténation |
| chaine + '.' | 'Chaîne de Kres.' | |
| chaine | 'Chaîne de Kres' | L'objet chaine non modifié par la concaténation |
| chaine = "Hello, World"<br>chaine | 'Hello, World' | Ce n'est pas une modification de chaine : c'est un autre objet. L'objet chaine initial a disparu. |

## 1- Le type chaine de caractères (suite)

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| chaine =str("Bon OK") | 'Bon OK' | C'est encore un autre objet. |
| | La concaténation | |
| phrase = "Hello" + "," + " " + "World" | 'Hello, World' | ***Nota*** : bcp d'allocation d'objets. |
| " " .join(("Bonjour", "la", "Terre")) | 'Bonjour la Terre' | + performant car peu d'allocations. |
| "-" .join(("Bonjour", "la", "Terre")) | 'Bonjour-la-Terre' | + performant car peu d'allocations. |
| | | |
| 'Dès l\'arrivée' | 'Dès l'arrivée' | Echappement |

## 1- Le type chaine de caractères : suite 1

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| texte = """L'objet est : "stable" enfin""" | | Triple guillemets. Pas besoin d'échapper les simples et doubles quotes. S'utilisent aussi pour le doc. |
| print(texte) | L'objet est : "stable" enfin | Affichage correct. |
| """ Insérer des sauts<br>... de ligne<br>... """ | ' Insérer des sauts<br>\nde ligne\n' | On peut insérer des sauts de ligne avec les triples quotes.<br><br>\n : instruction demandant à python d'insérer le saut de ligne. |

## 1- Le type chaine de caractères : suite 1

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| texte1 = """Insérer des saut<br>... de ligne<br>... par ligne<br>... """ | | Plaçons la chaine avec sauts dans une variable. |
| texte1 | 'Insérer des saut **\n**de ligne**\n**par ligne**\n**' | On voit bien les sauts de ligne. |
| print(texte1) | Insérer des saut<br>de ligne<br>par ligne | Les sauts de ligne sont préservés. |
| print("Insérer soi-même \n des sauts") | Insérer soi-même<br>des sauts | On peut mettre les sauts soi-même. |
| "Hello"[4] | 'Hell' | L'objet chaine est sliceable |

## 2- Plus loin sur les chaines de caractères

**Chaînes de caractères et listes (cf. chapitre sur les listes).**

Les chaînes de caractères peuvent être considérées comme des listes. Nous pouvons donc utiliser certaines propriétés des listes comme les tranches.

```
>>> animaux = "girafe tigre"
>>> animaux
'girafe tigre'
>>> len(animaux)
12
>>> animaux[3]
'a'
```

***Chaine comme liste***

```
>>> animaux = "girafe tigre"
>>> animaux[0:4]
'gira'
>>> animaux[9:]
'gre'
>>> animaux[:-2]
'girafe tig'
```

***Faire des tranches***

```
>>> animaux = "girafe tigre"
>>> animaux[4]
'f'
>>> animaux[4] = "F"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

***Contrairement aux listes , les chaines sont non modifiables***

## 2- Plus loin sur les chaines de caractères : suite

### Méthodes associées aux chaînes de caractères

Plusieurs méthodes sont associées au type chaine. On peut aussi importer string par import string et disposer d'autres constantes et méthodes.

*upper()*, *lower()*, *capitalize()* :
majuscule, minuscule, 1è lettre du mot en majuscule.

*split(), rsplit()* : renvoie une liste de mots, rsplit() commence à la fin de la chaine (reverse split)

```
>>> animaux = "girafe tigre singe"
>>> animaux.split()
['girafe', 'tigre', 'singe']
>>> for animal in animaux.split():
...     print animal
...
girafe
tigre
singe
>>> animaux = "girafe:tigre:singe"
>>> animaux.split(":")
['girafe', 'tigre', 'singe']
```

***Indiquer le séparateur***

```
>>> x = "girafe"
>>> x.upper()
'GIRAFE'
>>> x
'girafe'
>>> 'TIGRE'.lower()
'tigre'
>>> x[0].upper() + x[1:]
'Girafe'
>>> x.capitalize()
'Girafe'
```

***Quelques utilisations***

## 2- Plus loin sur les chaines de caractères : suite 1

### Méthodes associées aux chaînes de caractères

*find()* :
Recherche dans les chaines de caractères. Si l'élément recherché est trouvé, alors l'indice du début de l'élément dans la chaîne de  caractères est renvoyé. Si l'élément n'est pas trouvé, alors la valeur -1 est renvoyée.

```
>>> animal = "girafe"
>>> animal.find('i')
1
>>> animal.find('afe')
3
>>> animal.find('tig')
-1
>>> animaux = "girafe tigre"
>>> animaux.find("i")
1
```

Si l'élément recherché est trouvé plusieurs fois, seul l'indice de la première occurrence est retourné.

***Quelques utilisations***

## 2- Plus loin sur les chaines de caractères : suite 2

### Méthodes associées aux chaînes de caractères

*replace(), count()* :
Remplace une chaine, et fait compte le nombre d'occurrences d'une chaîne de caractères passée en argument respectivement.

```
>>> animaux = "girafe tigre"
>>> animaux.replace("tigre", "singe")
'girafe singe'
>>> animaux.replace("i", "o")
'gorafe togre'

>>> animaux = "girafe tigre"
>>> animaux.count("i")
2
>>> animaux.count("z")
0
>>> animaux.count("tigre")
1
```

***Quelques utilisations***

*str()* :
Convertit en chaine.

```
>>> i = 3
>>> str(i)
'3'
>>> i = '456'
```

## 2- Plus loin sur les chaines de caractères : suite 3

### Méthodes associées aux chaînes de caractères

*join()* :
convertit une liste de chaînes de caractères en une chaîne de caractères.
Attention, la fonction *join()* ne s'applique qu'à une liste de chaînes de caractères.

```
>>> seq = ["A", "T", "G", "A", "T"]
>>> seq
['A', 'T', 'G', 'A', 'T']
>>> "-".join(seq)
'A-T-G-A-T'
>>> " ".join(seq)
'A T G A T'
>>> "".join(seq)
'ATGAT'
```

***Utiliser join()***

```
>>> maliste = ["A", 5, "G"]
>>> " ".join(maliste)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 1: expected string, int found
```

***Nota :*** la fonction *join()* ne s'applique qu'à une liste de chaînes de caractères.

## 2- Plus loin sur les chaines de caractères : suite 4

### Méthodes associées aux chaînes de caractères

*capwords()* :
met la 1ère lettre des mots en majuscule. Contenu dans le module string.

```
>>> import string

>>> string.capwords(' hello    world! ')
'Hello World!'
```

*strip()*, *expandtabs()* :
gérer les espaces dans une chaîne en supprimant les blancs non significatifs ou en remplaçant les tabulations par un nombre fixe d'espaces.

```
>>> string.strip('  hello  world! \n ') # [2nd arg]
'hello  world!'
>>> string.expandtabs('\thello world!', 4)
'    hello world!'
```

## 2- Plus loin sur les chaines de caractères : suite 4

### Toutes les méthodes associées à un objet : dir()

Pour avoir une liste exhaustive de l'ensemble des méthodes associées à une variable particulière, vous pouvez utiliser la commande dir().

```
>>> animaux = "girafe tigre"
```

```
>>> dir(animaux)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__
'__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__str__', 'capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'isl
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition'
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split'
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

### Aide et documentation d'une fonction particulière

```
>>> help(animaux.split)
Help on built-in function split:

split(...)
```

## 2- Plus loin sur les chaines de caractères : suite 5

### Formater une chaine avec *format()*

La méthode *format()* permet une meilleure organisation de l'affichage des variables. Des accolades vides {} précisent l'endroit où le contenu de la variable doit être inséré.

```
>>> x = 32
>>> nom = "John"
>>> print('{} a {} ans'.format(nom, x))
John a 32 ans
```

```
# Trous avec ordre d'affichage
>>> x = 32
>>> nom = "John"
>>> print('{0} a {1} ans'.format(nom, x))
John a 32 ans
>>> print('{1} a {0} ans'.format(nom, x))
32 a John ans
>>> print('{x} a {nom:10} ans'.format(nom = nom, x = x))
32 a John ans
```

On indique dans les accolades {} dans quel ordre afficher les variables (0 pour la variable à afficher en premier, 1 pour la variable à afficher en second, etc)., ou en y écrivant le nom de la variable (dé-corrélation)

Largeur de la colonne d'affichage = 10 caractères.

## 2- Plus loin sur les chaines de caractères : suite 6

### Formater une chaine avec *format() :* précision

La méthode *format()* permet une meilleure organisation de l'affichage des variables. Des accolades vides {} précisent l'endroit où le contenu de la variable doit être inséré.

```
>>> poids = (4500.0 + 2575)/14800
>>> print("Le pois est:", poids)
Le poids est: 0.478040540541
>>> print("Le pois est: {:.3f}".format(poids))
Le poids est: 0.478
```

1. Les deux points (:) on veut préciser le format.
2. f indique qu'on veut afficher la variable sous forme d'un réel (float).
3. .3 indique la précision voulue, soit ici 3 chiffres après la virgule.

***Légende***

```
# formater des entiers
>>> x = 61
>>> print("L'élément a {:d} items".format(x))
L'élément a {:d} items
```

***d pour entier, s pour string***

## 2- Plus loin sur les chaines de caractères : suite 7

### Formater une chaine avec *format() :* alignements

Il est possible de préciser sur combien de caractères vous voulez qu'un résultat soit écrit et comment se fait l'alignement (à gauche, à droite ou centré). Dans la portion de code suivant, le caractère ; sert de séparateur entre les instructions sur une même ligne

```
# alignements
>>> print(10) ; print(1000)
10
1000
>>> print("{:>6d}".format(10) ; print "{:>6d}".format(1000))
  10
1000
>>> print("{:<6d}".format(10) ; print "{:<6d}".format(1000))
10
1000
>>> print("{:^6d}".format(10) ; print "{:^6d}".format(1000))
 10
1000
>>> print("{:*^6d}".format(10) ; print "{:*^6d}".format(1000))
**10**
*1000*
```

> : aligné à droite,
< : aligné à gauche
^ : centré

légende

** : caractère de remplissage*

## 2- Plus loin sur les chaines de caractères : suite 8

### Formater une chaine : ancienne méthode

Sur d'anciens codes Python, vous pourrez trouver l'écriture formatée suivante : la syntaxe est légèrement différente.

```
>>> x = 32
>>> nom = 'John'
>>> print "%s a %d ans" % (nom,x)
John  a  32  ans

>>> nbG = 4500
>>> nbC = 2575
>>> propGC = (4500.0 + 2575)/14800
>>> print "On a %d G et %d C -> prop GC = %.2f" % (nbG,nbC,propGC)
On a 4500 G et 2575 C -> prop GC = 0.48
```

% : l'endroit où sera placé la variable dans la chaine de caractères.

d : entier, i : entier, f : float

% : indique les variables à formater.

Précision de 2 chiffres après la virgule.

```
# Formatage à l'ancienne
from decimal import Decimal

def ligne(article, dte, total):
    print("%10s %d %.f"  % (article, qte, total))

ligne("Patates", 4, Decimal('1.50'))
ligne("Poire", 2, Decimal('0.45'))
```

## 2- Plus loin sur les chaines de caractères : suite 9

### Affichage avec pprint

Le module pprint permet de rendre l'affichage plus joli.
Adapté pour afficher rapidement des structures Python comme les listes et dictionnaire, et pour le débogage.

```
# Afficher des structures Python
# pprint
from pprint import pprint


maliste = [
    {
        'nom':'Carotte',
        'quantite':3,
        'total':1.5
    },
    {
        'nom':'Fraise',
        'quantite':1,
        'total':0.25
    }
]

pprint(maliste)
```

```
[{'nom': 'Carotte', 'quantite': 3, 'total': 1.5},
 {'nom': 'Fraise', 'quantite': 1, 'total': 0.25}]
```

*Affichage avec* ***pprint***

```
[{'total': 1.5, 'nom': 'Carotte', 'quantite': 3}, {'total': 0.25, 'nom': 'Fraise', 'quantite': 1}]
```

*Affichage avec* ***print***

## *1-* Le type booléen

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| True | True | |
| False | False | |
| type(True) | bool | |
| ok = True | | Une variable |
| ok | True | |
| isinstance(ok, bool) | True | |
| ok = bool(True) | | |
| ok | True | |

## 1- Le type booléen (suite)

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| Opérations booléennes | | |
| not True | False | |
| True and False | False | |
| False and True | False | |
| 3 > 1 and 7 > 5 | True | |
| True or False | True | Ou inclusif |
| qte = 15 | | |
| qte > 7 | True | |
| qte == 15 | True | |

## 1- Le type None

Ce type signifie rien.

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| type(None) | <class 'NoneType'> | Rien |
| None == False | False | None n'est pas un booléen |
| None == True | False | |
| retour = print('Coucou') | Coucou | |
| type(retour) | <class 'NoneType'> | print retourne None, donc rien. |
| None | | Rien ne se passe sur le shell |
| | | |

## 1- Le type liste (ou tableaux)

*list(), append(), remove()*

- Ce type est un tableau modifiable (mutable) et itérable (iterable).
- Peut contenir des éléments de types différents
- Les éléments sont définis par leur indice.
- Les indices commencent à 0.

| 3 | -2 | Jardin | 2.0 |
|---|---|---|---|

Indice 0

## 1- Le type liste (ou tableaux) : suite

*list(), append(), remove()*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| liste1 = list() | | Création avec le constructeur. |
| liste1 | () | |
| liste2 = [] | | Création en utilisant des crochets |
| liste2 | () | |
| liste3 = [5, 'Bonjour'] | | Remplissage à la création |
| liste3 | [5, 'Bonjour'] | |
| liste3.append('le monde') | | Ajouter un élément |
| liste3 | [5, 'Bonjour', 'le monde'] | L'ordre est conservé |

## 1- Le type liste (ou tableaux) : suite 1

*list(), append(), remove()*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| liste3.*remove(*0*)* | | L'argument ce n'est pas la clé, mais la valeur stockée |
| liste3.*remove(*5*)* | | |
| liste3 | ['Bonjour', 'le monde'] | 5 a été enlevé |
| liste3.*remove(*'Bonjour'*)* | | |
| liste3 | ['le monde'] | |
| liste3.*remove(*'Le monde'*)* | | Il faut respecter la casse |
| liste3.*remove(*'le monde'*)* | | |
| liste3 | [] | |

## 1- Le type liste (ou tableaux) : suite 2

*list(), append(), remove()*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| liste2 = ['Bonjour'] | | |
| liste3 = ['le monde'] | | |
| liste4 = liste2 + liste3 | | On peut concaténer des listes |
| liste4 | ['Bonjour', 'le monde'] | |
| | Les listes sont ordonnés, on peut écrire ceci | |
| [3]*5 | [3, 3, 3, 3, 3] | On a une liste de 5 éléments |
| maliste = [6]*4 | | Créer une grande liste rapidement |
| maliste | [6, 6, 6, 6] | |

## *1-* Le type tuple (ou tableaux non modifiables)

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| tuple1 = tuple() | | Utiliser le constructeur (tuple vide) |
| tuple1 | () | |
| tuple2 = () | | Création avec des parenthèses. Ici un autre tuple vide. |
| tuple2 | () | |
| tuple3 = (1, 'Bonjour') | | Création avec remplissage |
| tuple3 | (1, 'Bonjour') | |
| tuple3.*append(*'Monde'*)* | | KO : un tuple est non modifiable, il faut en créer un autre. |
| tuple3 = tuple3 + ('Monde') | | Il manque une virgule après l'élément à ajouter. |

## *1-* Le type tuple (ou tableaux non modifiables) : suite

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| tuple3 = tuple3 + ('Monde') | | Il manque une virgule après l'élément à ajouter. |
| tuple3 = tuple3 + ('Monde',) | | La virgule est nécessaire : l'ancien tuple3 a été supprimé, recyclé par le système. |
| tuple3 | (1, 'Bonjour', 'Monde') | Création avec des parenthèses. Ici un autre tuple vide. |
| Les tuples sont ordonnés, on peut écrire ceci | | |
| (3,)*5 | (3, 3, 3, 3, 3) | On a un tuple de 5 éléments |
| tuple4 = (7,)*4 | | Créer un grand tuple rapidement |
| tuple4 | (7, 7, 7, 7) | |

## *1-* Le type tuple (ou tableaux non modifiables) : suite 1

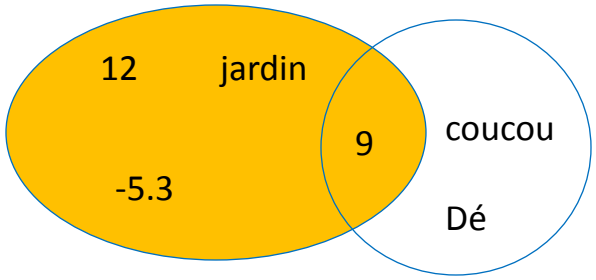
**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| (1,) + [5] | | On ne peut mélanger ces types. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 1- Le type set (ou ensembles)

*add(), union(), difference(), intersection() , isdisjoint() , issubset() , issuperset(), etc.*

- Ce type est un ensemble(comme en mathématiques) .
- Peut contenir des éléments de types différents
- Il n'y a pas d'ordre pour les éléments
- Les doublons sont impossibles (-> fusion/dédoublonner des listes)
- Plusieurs opérations disponibles : union, intersection, etc.



## 1- Le type set (ou ensembles) : suite

*add(), union(), difference(), intersection() , isdisjoint() , issubset() , issuperset(), etc.*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| ensemble = set() | | Créer en utilisant le constructeur |
| ensemble | set() | |
| ensemble2 = {} | | Créer en utilisant des accolades |
| ensemble2 | {} | Ne pas confondre avec dictionnaires |
| ensemble2 = {1, 2, 3} | | Remplir |
| ensemble2 | {1, 2, 3} | |
| ensemble3 = {"Hello" , "World"} | | |
| ensemble3 | {'Hello' , 'World'} | |

## 1- Le type set (ou ensembles) : suite 1

*add(), union(), difference(), intersection() , isdisjoint() , issubset() , issuperset(), etc.*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| ensemble3.*add(*1*)* | | Ajouter un elmt avec *add(.)* |
| ensemble3 | {'Hello' , 1, 'World'} | L'ordre n'est pas gardé. |
| ensemble3.*add(*"World"*)* | | |
| ensemble3 | {'Hello' , 1, 'World'} | Unicité des éléments préservée |
| ensemble4 = set("Hello" ) | | |
| ensemble4 | {'H' , 'e', 'l' , 'l', 'o'} | Une chaine est itérable |
| ensemble5 = set(15 ) | | |
| ensemble5 | TypeError: 'int' object is not iterable | |

## 1- Le type set (ou ensembles) : suite 2

*add(), union(), difference(), intersection() , isdisjoint() , issubset() , issuperset(), etc.*

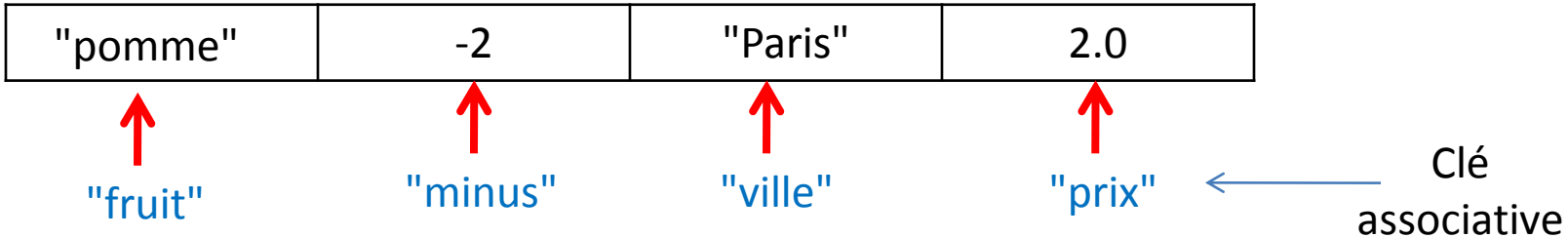
**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| ensemble1 = {1, 2} | | |
| ensemble2 = {5, 2} | | |
| ensemble = ensemble1.*union(*ensemble2*)* | | Union de 2 ensembles |
| ensemble | {1, 2, 5} | |
| ensemble3 = ensemble1.*intersection(*ensemble2*)* | | Intersection |
| ensemble3 | {2} | |
| ensemble4 = ensemble1.*difference(*ensemble2*)* | | Différence |
| ensemble4 | {1} | |

## 1- Le type dict (ou dictionnaire)

*dict(), get(), items(), keys() , pop() , popitem() , values(), update(), etc.*

- Ce type est une structure de type clé/valeur .
- Permet l'accès rapide aux informations connues
- On ne peut avoir de doublons sur la clé
- Les éléments stockés peuvent être de types différents
- L'accès se fait via la clé.



## 1- Le type dict (ou dictionnaire) : suite

*dict(), get(), items(), keys() , pop() , popitem() , values(), update(), etc.*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| dict() | | |
| dico = dict() | {} | Utiliser le constructeur |
| dico = {} | | Utiliser les accolades |
| dico1 = *dict(*(("hello", "bonjour"),<br>("world", "monde"))*)* | | Notation complexe |
| dico1 | {'hello': 'bonjour', 'world': 'monde'} | |
| dico2 = {'hello': 'bonjour', 'world': 'monde'} | | Meilleure notation |
| dico1 == dico2 | True | Les 2 sont égaux |

## 1- Le type dict (ou dictionnaire) : suite 1

*dict(), get(), items(), keys() , pop() , popitem() , values(), update(), etc.*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| dico1["hello"] | 'bonjour' | Atteindre un élément |
| dico1["blue"] = "bleu" | | Ajout d'une clé |
| dico1 | {'hello': 'bonjour', 'world': 'monde', 'blue': 'bleu'} | |
| dico1["blue"] = "bleue" | | Modifier une valeur |
| dico1 | {'hello': 'bonjour', 'world': 'monde', 'blue': 'bleue'} | |
| type(dico1) | <class 'dict'> | |

## 1- Le type dict (ou dictionnaire) : suite 2

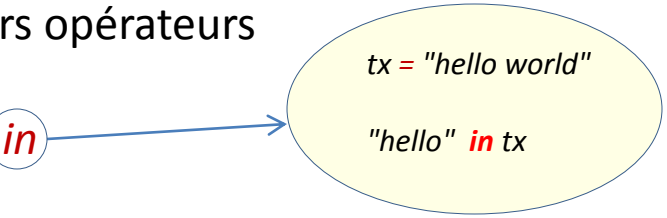
*dict(), get(), items(), keys() , pop() , popitem() , values(), update(), etc.*

**Travaux pratiques :** Tapez ces instructions dans la console.

| Instruction | Sortie | Commentaire |
|---|---|---|
| dico1.*get(*"hello"*)* | 'bonjour' | Atteindre un élément |
| dico1.*keys()* | | Donne les clés |
| dico1.*pop(*"hello"*)* | | Supprime l'élément selon clé |
| dico1 | {'world': 'monde', 'blue': 'bleue'} | |
| dico1.*values()* | 'monde', 'bleue' | Donne les valeurs |
| dico1.*items()* | | Affiche les items |
| dico1.*popitem()* | | Supprime les items |
| dico1 | {} | Il n'y a plus rien |

## 1- Les opérateurs logiques, la fonction input()

- Les opérateurs logiques permettent d'écrire les structures logiques
- Conditions complexes possibles avec plusieurs opérateurs
- Opérateurs : *>, >=, <, <=, ==, !=, or, and, not, in*




| A | B | C |
|---|---|---|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

*or : table de vérité*

| A | B | C |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

*and : table de vérité*

## 2- La fonction input() pour récupérer les saisies

- Cette fonction attends la saisie de l'utilisateur
- Créer toujours une variable pour récupérer cette saisie
- La saisie est une chaine : on peut la convertir en entier, flottant, etc.

### Exemple

| Instruction | Sortie | Commentaire |
|---|---|---|
| nom = input("Quel est votre nom? ") | Quel est votre nom? | Si on saisit **Dupont** |
| print(nom) | Dupont | |
| entree = input("Quel est le score? ") | Quel est le score? | Si on saisit **42** |
| score = int(entree) | | |
| type(score) | <class 'int'> | |

## 3- Les structures de décision

- On teste des conditions logiques
- Le résultat de tests est un booléen (True ou False)

```
if [condition1]:
⟷ insts
elif [condition2]:
⟷ insts
elif [condition3]:
⟷ insts
else [condition4]:
⟷ insts
```

*Syntaxe*

```
def comparer(x,y):
    if x == y:
        return "Les deux nombres sont égaux"
    elif x < y:
        return str(x) + " est inférieur à " + str(y)
    else:
        return str(x) + " est supérieur à " + str(y)

print(comparer(3,2))
```

*exemple*

⟷ *Votre indentation sera de 4 espaces*

## 3- Les structures de décision : suite

- Il est possible d'imbriquer des if
- dans chaque bloc on peut avoir 1 ou plusieurs instructions

```
if [condition1]:
    insts
elif [condition2]:
    insts
elif [condition3]:
    insts
else [condition4]:
    insts
```

*Syntaxe*

```
def comparer(x,y):
    if x == y:
        return "Les deux nombres sont égaux"
    elif x < y:
        if x*10 < y:
            return str(x) + " est très inférieur à " + str(y)
        return str(x) + " est inférieur à " + str(y)
    else:
        return str(x) + " est supérieur à " + str(y)

print(comparer(2,400))
```

*if imbriqué*

*Votre indentation sera de 4 espaces*

## 4- La boucle while (tant que)

- Quasi identique à la boucle dans les plusieurs langages (PHP, Java, etc.)
- Exécute des instructions tant qu'une condition est vraie

```
while [condition1]:
    inst1
    inst2
    inst3
```

*Syntaxe*

```
i = 0
while i < 5:
    print(i)
    i += 1
print(i)
```

*initialiser*

*modifier la condition*

*Exemple*

## Nota

- Initialiser une variable pour entrer dans la boucle
- Attention aux boucles infinies (CTRL+C termine le programme)
- Utiles si on ne sait pas combien de fois boucler.

## 5- La boucle for (pour)

- Assez différente  en Python par rapport aux autres langages.
- Associé en général avec le générateur *range(start,stop,step)* qui génère une séquence de nombres.
- Utilise l'opérateur in.
- On peut itérer sur tout élément itérable (liste, chaine, etc.)
- Le nombre d'itérations est souvent connu

```
for  variable for  [itérable]:
        inst1
        inst2
        inst3
```

*Syntaxe*

## 5- La boucle for (pour) : suite

**Associé à range()**

```
# avec range(5), on génerera de 0 à 4
for i in range(5):
    print(i)
```

*Exemple avec range()*

**On peut générer une liste dans un print et filtrer dedans.**

- permet d'écrire du code court mais …
- plus difficile à maintenir que un for classique
- la sortie est une liste (on obtient une liste)

```
# la sortie est une liste (on obtient une liste)
print([i for i in range(5)])
```

*Exemple for dans un print() (cf. liste en intension)*

## 5- La boucle for (pour) : suite 1

**Filtrer dans un print**

On peut filtrer dans le print

*Nota :* ne pas abuser de cette écriture, sinon le code devient illisible.

```
#on peut mettre des conditions dans la liste pour filtrer les valeurs ()
print([i for i in range(5) if i%2])     #nombres impairs
print([i for i in range(5) if not i%2]) #nombres pairs
```

*Filtrage de la liste donnée par for*

## 6- La boucle with (avec) : lecture /écriture d'un fichier

Cette boucle permet de lire un fichier rapidement.
En quittant with, le fichier est fermé automatiquement de façon propre.

```
#Afficher le fichier
#en quittant with, le fichier va être fermé de façon propre (ne peut plus être lu)
with open("main.py", "r") as fichier:
    print(fichier.read())

print("Bonjour") #la sortie sera juste Bonjour, car le fichier a été fermé par le with
print(fichier.read()) #ligne inutile car le fichier a été fermé


#Ecrire dans un fichier puis le lire
with open("sortie", "w") as fichier:
    print(fichier.write("Bonjour"))

with open("sortie", "r") as fichier:
    print(fichier.read())
```

*Exemple : lire /écrire dans un fichier avec with*