

FASCICULE DE COURS

PROGRAMMATION PYTHON

1^{er} Génie Informatique

Semestre 1 ♣

PRÉSENTÉ PAR

✓ **Dr. Taoufik BEN ABDALLAH**

✉ taoufik.benabdallah@iit.ens.tn

✓ **Dr. Rania REBAI BOUKHRISS**

✉ rania.rebai@iit.ens.tn

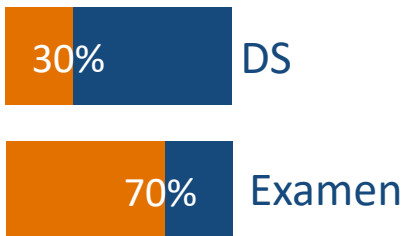


Introduire les concepts de base
du langage Python_version **3**



Cours \pm **15h** ☒

TP \pm **20h** ☒



Contexte : **Langages de programmation**



Polarité des langages de programmation

Language Ranking: IEEE Spectrum

| Rank | Language | Type | Score |
|------|------------|-----------|-------|
| 1 | Python | 🌐 🖥️ ⚙️ | 100.0 |
| 2 | Java | 🌐 📱 🖥️ | 95.4 |
| 3 | C | 📱 🖥️ ⚙️ | 94.7 |
| 4 | C++ | 📱 🖥️ ⚙️ | 92.4 |
| 5 | JavaScript | 🌐 | 88.1 |
| 6 | C# | 🌐 📱 🖥️ ⚙️ | 82.4 |
| 7 | R | 🖥️ | 81.7 |
| 8 | Go | 🌐 🖥️ | 77.7 |
| 9 | HTML | 🌐 | 75.4 |
| 10 | Swift | 📱 🖥️ | 70.4 |

Python conforte sa place de leader

Classement des langages de programmation selon IEEE, 2021



PLAN

DU COURS

Plan du cours: PYTHON BASICS

1 Introduction sur le langage Python

2 Conteneurs standards en Python

3 Fonctions & Générateurs

4 Gestion de Fichiers

5 Gestion des Exceptions



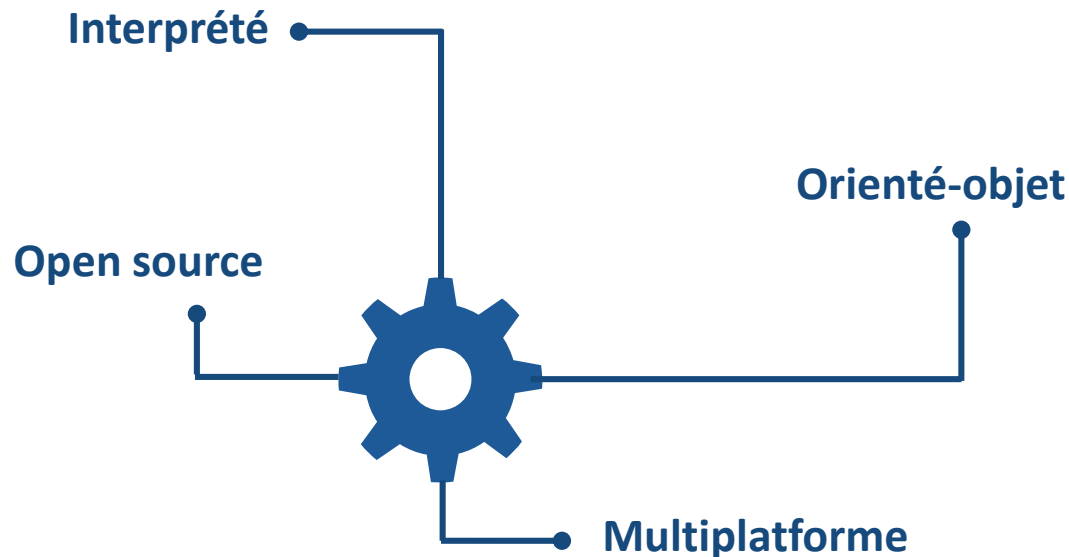
CHAPITRE 1

INTRODUCTION SUR LE LANGAGE PYTHON

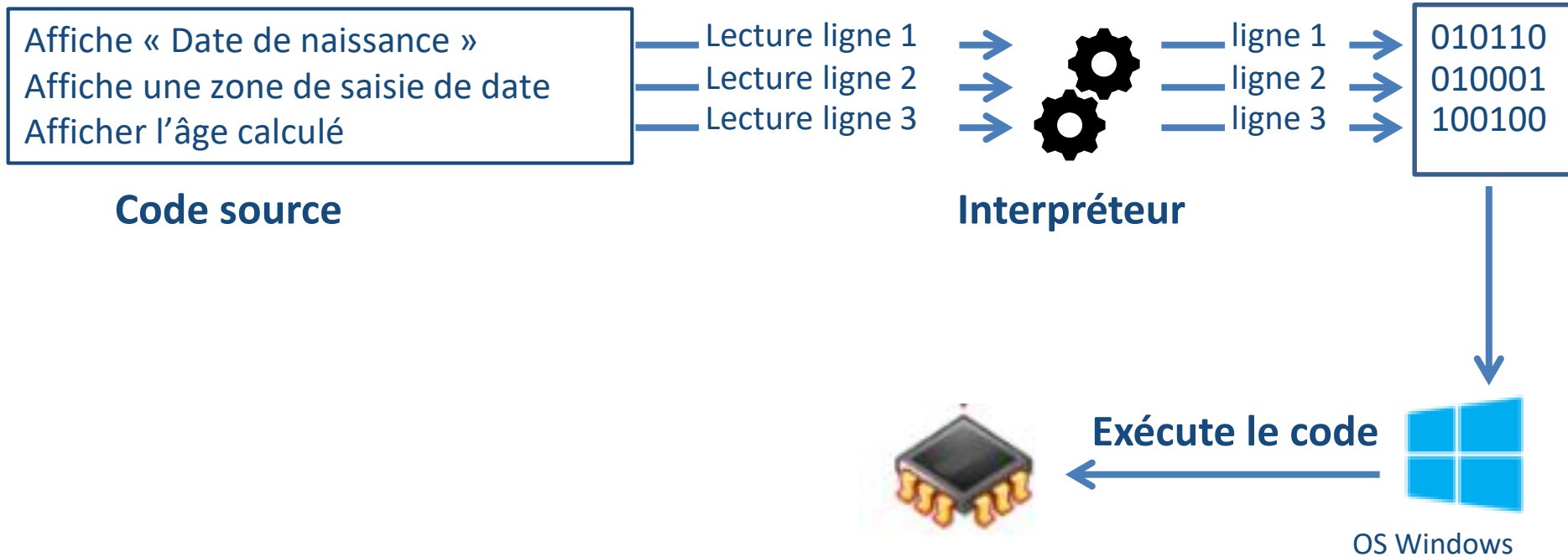


C'est quoi Python?

- Langage de programmation de haut niveau inventé en 1991 par Guido van Rossum
- Utilisé plus généralement en analyse de données et en calcul scientifique



Python : langage interprété



Pourquoi apprendre Python?

Multi-fonctions

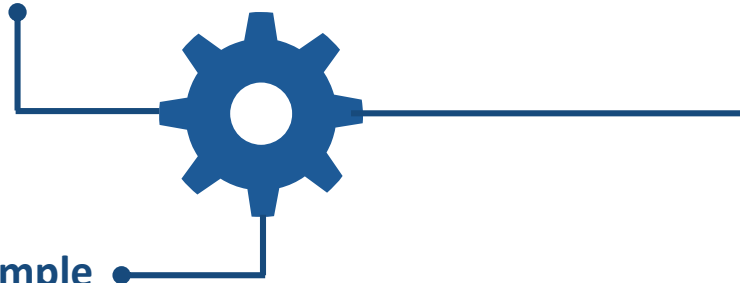
- 💡 Data science
- 💡 Développement Web

Populaire

- 💡 Version3

Simple

- 💡 Typage dynamique
- 💡 Syntaxe simple



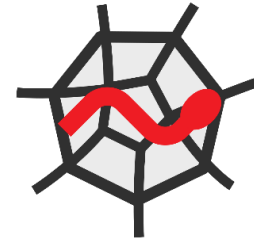
Environnement de développement (IDE)



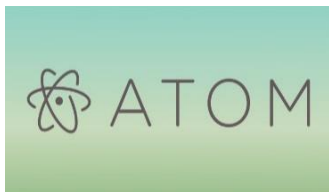
IDLE



PyCharm



Spyder



Atom




PyDev



Jupyter Notebook

Jupyter Notebook

 **Jupyter Notebook** ne fonctionne pas uniquement comme un IDE, mais aussi comme un outil de présentation

- ❑ permettant le travail en mode interactif ou fichier

"Jupyter Notebook should be an integral part of any Python data scientist's toolbox. It's great for prototyping and sharing notebooks with visualizations"

Anaconda?

☰ **Anaconda** est une **distribution complète** de Python

- ❑ Éviter les problèmes d'incompatibilités entre les différents packages
- ❑ Proposer un outil de gestion de packages appelé **Conda** utile pour mettre à jour et installer facilement les librairies destinées au **calcul numérique** (numpy, scipy, panda, matplotlib, etc.)

Anaconda = Python + librairies + **Jupyter notebook**



Installation d'Anaconda

 Lien de téléchargement : <https://www.anaconda.com/distribution/>



Individual Edition

Your data science toolkit

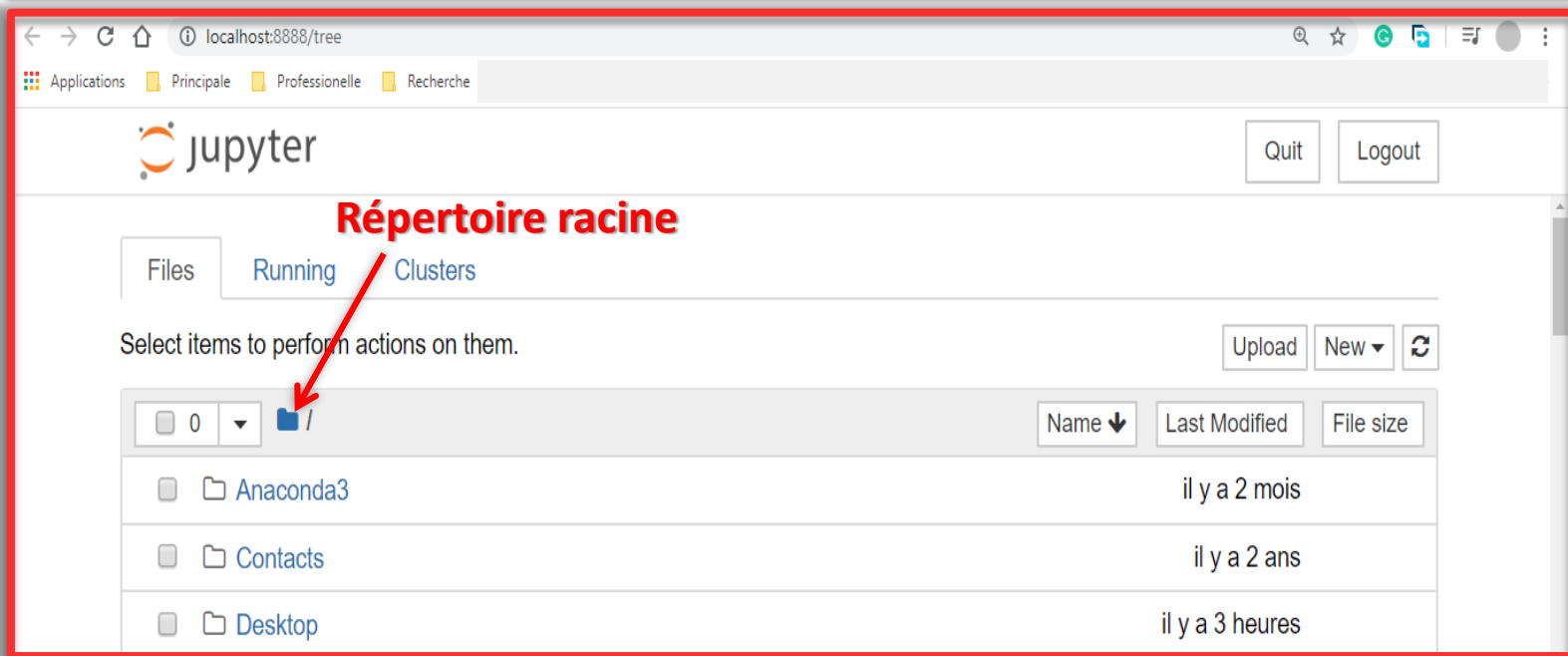
With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.



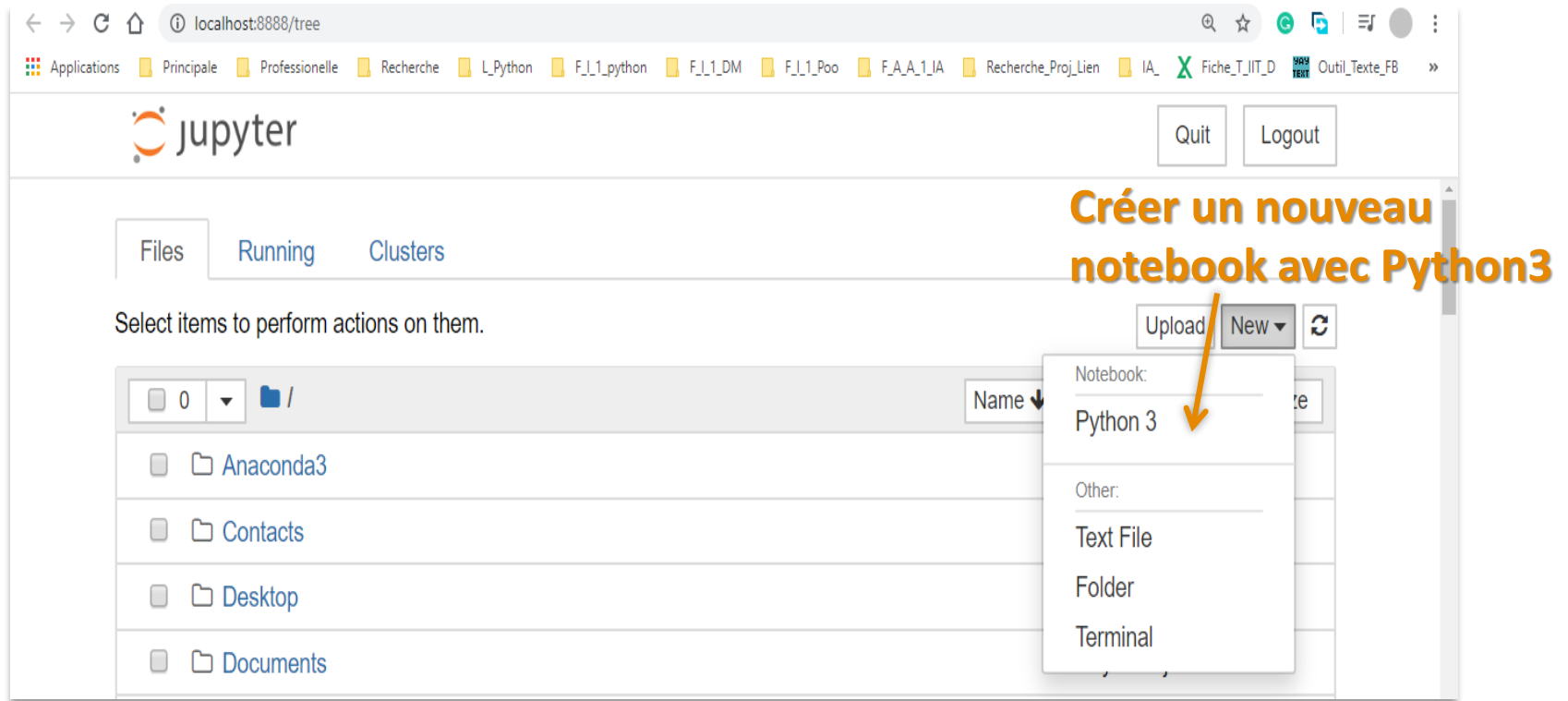
Anaconda Prompt : Ouvrir Jupyter Notebook

```
(base) C:\Users\ASUS> D:
```

```
base) D:\>jupyter notebook
```



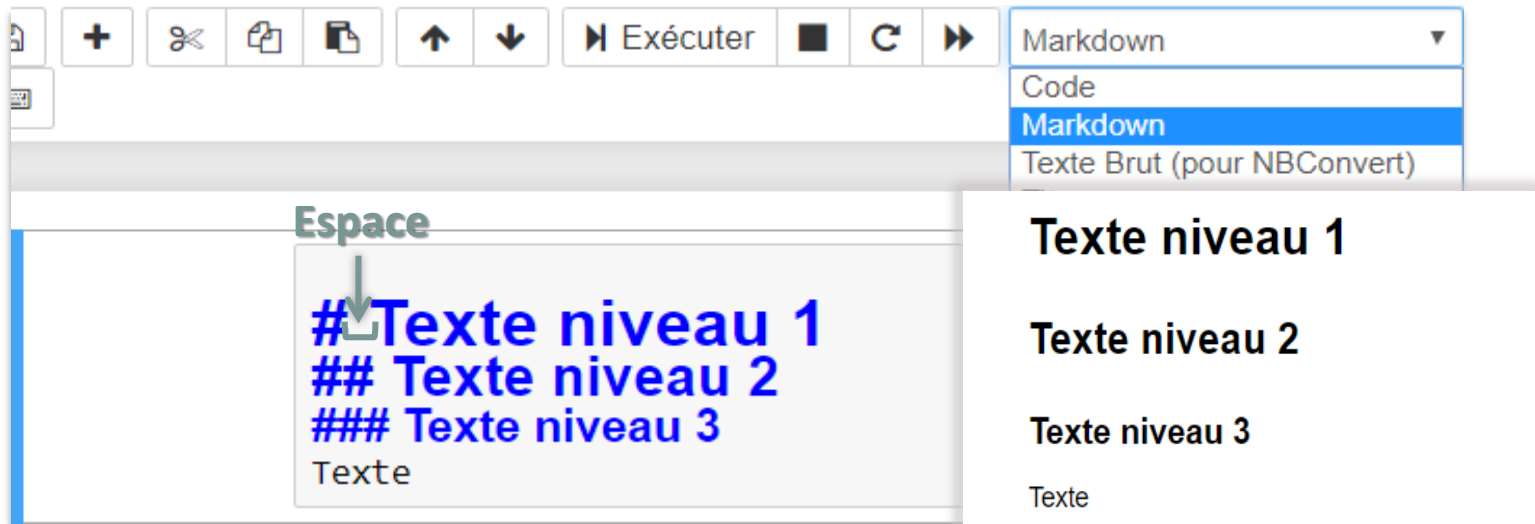
Création d'un nouveau Notebook



Le fichier résultat de notebook est d'extension **.ipynb**

Format Markdown

- Markdown : langage de balisage léger
- Rédiger du **texte formaté** (gras, italique, liens, titres, images, formules mathématiques, etc.) avec quelques balises très simples



NB. Un document Markdown peut être lu sans donner l'impression d'avoir été balisé ou formaté par des instructions particulières

Commentaire



Un commentaire serve à :

- ❑ Expliquer ce qui se passe dans une portion de code
- ❑ Documenter qui a écrit le code ou d'autres informations
- ❑ Désactiver une ligne de code

```
# commentaire sur une seule ligne
```

Commentaire sur un
seul ligne #

```
"""
```

```
commentaire sur plusieurs lignes
```

```
"""
```

Commentaire sur
plusieurs lignes """

```
'\ncommentaire sur plusieurs lignes\n'
```

?

Variables

- Une variable est caractérisée par un **identificateur (nom)**

- Et un **type** automatiquement attribué

- 💡 Le nom des variables peut être constitué de lettres minuscules (**a à z**), lettres majuscule (**A à Z**), nombres (**0 à 9**) et/ou caractère souligné (**_**)

NB.

Python est sensible à la casse, ce qui signifie que les variables « test », « Test » ou « TEST » sont différentes

→ N'utilise pas d'espace dans un nom de variable & des mots réservés

Mots réservés

```
import keyword
for e in keyword.kwlist:
    print (e, end="\t")

print ("")
print ("Le nombre de mots réservés est: ", len(keyword.kwlist))
```

| | | | | | | | | | | | | |
|-------|----------|------|--------|---------|--------|-------|--------|-------|--------|----------|-------|-------|
| False | None | True | and | as | assert | async | await | break | class | continue | def | |
| del | elif | else | except | finally | for | from | global | if | import | in | is | lambd |
| a | nonlocal | | not | or | pass | raise | return | try | while | with | yield | |

↑
35 mots réservés

Type des variables

`type(var)` renvoie le type de la variable en argument `var`

`id(var)` renvoie un entier, représentant l'identifiant interne de la variable en argument `var` quel que soit son type

```
x=False  
type(x) # bool
```

```
id(x) # 504140000
```

```
x=1  
type(x) # int
```

```
id(x) # 504506560
```

Opérateurs

Les opérateurs arithmétiques

| | |
|-------------|---------------------------|
| x+y | Addition |
| x-y | soustraction |
| x*y | Multiplication |
| x/y | Division réelle |
| x**n | Élévation à la puissance |
| x//y | Division entière |
| x%y | Reste de division entière |
| -x | Opposé - opérateur unaire |

Les opérateurs de comparaison

| | |
|-------------------|--|
| x<y | Strictement inférieur |
| x<=y | Inférieur ou égal |
| x>y | Strictement supérieur |
| x>=y | Supérieur ou égal |
| x==y | Égal |
| x!=y | Différent |
| x is y | x et y représentent le même objet |
| x is not y | x et y ne représentent pas le même objet |

Les opérateurs logiques


| | |
|----------------|---------------------|
| x or y | Ou logique |
| x and y | Et logique |
| not x | Non logique |
| x ^ y | Ou Exclusif logique |

Les opérateurs bit à bit

| | |
|---------------------|---|
| x y | Ou bit à bit |
| x & y | Et bit à bit |
| x ^ y | Ou Exclusif bit à bit |
| x << y | Décalage de y bits vers la gauche sur x |
| x >> y | Décalage de y bits vers la droite sur x |

Opérateurs

Priorité des opérateurs



| | |
|---|--|
| ** | puissance |
| *, /, //, % | multiplication, division, division entière, reste |
| +, - | addition et soustraction |
| <<, >> | décalages bit à bit |
| &, ^, | Et bit à bit, Ou Exclusif bit à bit , Ou bit à bit |
| is, is not | Tests d'appartenance |
| <, <=, >, >=, !=, == | Comparaisons |
| or, and, not x, ^ y | Opérateurs logiques |

Les priorités des opérateurs de la plus haute à la plus basse

Affectation parallèle & multiple

```
x=8; y=8; z=12
```

```
x, y, z=8, 8, 12 ← Affectation parallèle  
print(x, y, z)
```

```
x=y=8 ; z=12 ← Affectation multiple  
print(x, y, z)
```

NB. L'ordre d'affectation est important

Activité : Étant donné **x= 11** et **y= 22**, écrire dans une seule ligne l'instruction qui permet de permuter la valeur de **x** par **y**. Afficher les nouvelles valeurs de **x** et **y**

```
x,y=11,22  
x,y=y,x  
print(x,y)
```


Instruction de sortie **print()**

 Affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne

 Ajoute un retour à la ligne par défaut

```
print("Bonjour") # Bonjour  
print("Taoufik") # Taoufik
```

```
print("Bonjour", end="")  
print("Taoufik") # BonjourTaoufik
```

Argument qui supprime le retour à la ligne et le remplace par la chaîne en entrée

```
print("Bonjour", "Taoufik", sep="-")  
print("Taoufik") # Bonjour-Taoufik
```

Argument qui met un séparateur entre les valeurs des autres arguments donnés en entrée

Instruction d'entrée **input()**

- Provoque une interruption dans le programme courant où l'utilisateur est invité à entrer des données au clavier

Entrée [*]:

```
chaine = input()
age = input("Age : ")

print("La valeur saisie de chaine : ", chaine)
print("La valeur saisie d'Age : ", age)
```



Taoufik



Age :

...

```
type(age) # str
```

Conversion de types

☰ Chaîne de caractères (str) → entier (int) : `int("chaîne")`

```
annN = input("Année de naissance :")
age = 2021-int(annN) # 1991
print("Age : ", age)
type(age) # int
```

☰ Chaîne de caractères (str) → type adéquat : `eval("chaîne")`

```
age = eval(input("Age : ")) # 30
print("La valeur saisie d'Age : ", age)
type(age) # int
```

```
a=eval("12.5")
type(a) # float
```

`a=int("12.5")?`

Conversion de types

☰ Entier (int) → réel (float) : **float()**

☰ Entier (int), réel (float) → chaîne de caractères : **str()**

```
a=33 # <class 'int'>
```

```
b="a35" # <class 'str'>
```

```
c=111.94 # <class 'float'>
```

```
d=112 # <class 'int'>
```

```
a=float(a) # ?
```

```
b=int(b) # ?
```

```
b=int(c) # ?
```

```
d=str(d) # ?
```

Écriture formatée **Ancienne méthode**

```
x = 29
nom = "Taoufik"
print("%s a %d ans" % (nom, x)) # Taoufik a 29 ans
```

The diagram illustrates the old Python formatting style. In the code, `%s` is boxed in blue and labeled 'str' below it. `%d` is boxed in green and labeled 'int' above it. A green arrow points from the `%d` box to the variable `x` in the tuple. A blue arrow points from the `%s` box to the variable `nom` in the tuple. A red circle highlights the `%` operator.

```
y=12.658456
print("y=%f"%y) # y=12.658456
```

The diagram shows the old Python formatting style for floats. `%f` is boxed in blue and labeled 'float' below it. A red circle highlights the `%` operator.

```
print("y=%.2f"%y) # y=12.66
```

float (avec deux chiffres après la virgule)

Conseil : Eviter cette écriture. Utiliser plutôt la fonction `ch.format()`

Écriture formatée `ch.format()`

- Permet une meilleure organisation de l'affichage des variables
- Agit sur la chaîne de caractères `ch` à laquelle elle est attachée par le point

NB. Les accolades `{ }` précisent l'endroit où le contenu de la variable doit être inséré

```
x=28; y=35.5; nom="x"
print("La valeur de {} est {}".format(nom,x))  #sans indiciage
print("{1} est la valeur de {0}".format(nom,x)) #Avec indiciage
print("{} est la valeur de {nomX}".format(x,nomX=nom))
print("La valeur de y est {:.2f}".format(y)) #Avec spécification de type
print("{:e}".format(12)) #Notation exponentielle
```

La valeur de x est 28
28 est la valeur de x
28 est la valeur de x
La valeur de y est 35.00
1.200000e+01

Quiz

1. Python est un langage (une seule réponse)

- ☒ interprété
- ☐ machine
- ☐ compilé
- ☐ binaire

2. Python est sensible à la casse (une seule réponse)

- ☒ Vrai
- ☐ Faux

3. Soit `n=12.0/4`; `x=12.0//5`; `m="13.0"` (Choix multiple)

- ☒ `type(n)` retourne *float*
- ☒ `type(x)` retourne *float*
- ☐ `eval(m)` convertie `m` en *int*
- ☒ `eval(m)` convertie `m` en *float*

■ CONTRÔLE DE FLUX



Exécution conditionnelle

💡 Test

`if <expression> :` ← **N'oubliez pas le double point**
tabulation
↔ `Traitement` **l'indentation (tabulation) !**

💡 Test à plusieurs cas

| | |
|--------------------------------------|--|
| <code>if <expression> :</code> | <code>if <expression> :</code> |
| ↔ <code>Traitement</code> | ↔ <code>Traitement</code> |
| <code>else :</code> | <code>elif <expression> :</code> |
| ↔ <code>Traitement</code> | ↔ <code>Traitement</code> |
| | <code>else :</code> |
| | ↔ <code>Traitement</code> |

NB. Une expression peut représenter des conditions assemblées avec les opérateurs logiques

Instructions conditionnelles imbriquées

Il est possible d'imbriquer des instructions conditionnelles les unes dans les autres, de manière à réaliser des structures de décision complexes

Exemple

```
x=eval(input())
y,z,a="KK","AA13", 10
if x==12:
    if y=="KK":
        if z=="AA13":
            print("T1")
        else:
            print("T2")
    else:
        print("T3")
else:
    print("T4")
```

L'instruction while

while <expression> :
tabulation
↔ Instructions à répéter

Si la condition est **fausse** au départ, le corps de la boucle n'est jamais exécuté

Si la condition est **vraie**, alors le corps de la boucle est répété *indéfiniment*

NB.

La boucle doit contenir au moins une instruction

Cette instruction doit changer la valeur d'une variable intervenant dans la condition, de manière à ce que cette condition puisse devenir fausse

n=2

```
while n>1:
    print("Bonjour!")
```

**Boucle
infinie**

L'instruction for

for <element> in <sequence> :

tabulation

↔ Instructions à répéter

Elle répète une séquence d'instructions autant de fois que le nombre d'éléments dans une **séquence** (**Conteneur** ou **Intervalle**)

💡 **Conteneur** : chaîne de caractère, liste, tuple, ensemble, ou dictionnaire

💡 **Intervalle** : **range**

range(val) : Énumère les entiers de **0** → **val-1**

range(val1, val2) : Énumère les entiers de **val1** → **val2-1**

range(val1, val2, val3) : Énumère les entiers de **val1** → **val2-1** par **pas** de **val3**

L'instruction for

```
for x in range(3):  
    print(x, end="/") # 0/1/2/
```

```
for x in range(2,6):  
    print(x, end="/") # 2/3/4/5/
```

```
for x in range(2,6,2):  
    print(x, end="/") # 2/4/
```

```
for x in range(6,2,-1):  
    print(x, end="/") # 6/5/4/3/
```

```
for x in range(4):  
    print(x, end="/") # 0/1/2/3/
```

```
x += 2
```



x prend les différentes valeurs imposées par range, même si elle est modifiée dans le corps de la boucle for

break & continue

break : sort violemment de la boucle et passe à la suite

```
for x in range(4):  
    if x==2:  
        break  
    print(x, end="/")  
print("YY")  
# 0/1/YY
```

continue : saute directement à l'itération suivante sans exécuter la suite du bloc d'instructions de la boucle

```
for x in range(4):  
    if x==2:  
        continue  
    print(x, end="/")  
print("YY")  
# 0/1/3/YY
```

Bloc d'instructions en fin de boucle

while <expression> :
 Instructions à répéter
else :
 Si sortie propre

for <element> **in** <sequence> :
 Instructions à répéter
else :
 Si sortie propre

NB.

On peut ajouter un bloc d'instructions en fin de boucle, qui sera réalisé si la sortie de la boucle s'est faite proprement (**sans break**), avec l'instruction **else**

Quiz

1. Quelle est la sortie du code suivant ? (une seule réponse)

```
x=True; y=z=False
if not x or y:
    print (x)
elif not x or not y and z:
    print (not x)
else: print ("None")
```

- ☐ True
- ☐ False
- ☒ None

2. Par quoi remplacer le vide pour que le code affiche **13AA** ? (une seule réponse)

```
for i in range(1,4):
    if i==5//2:
```

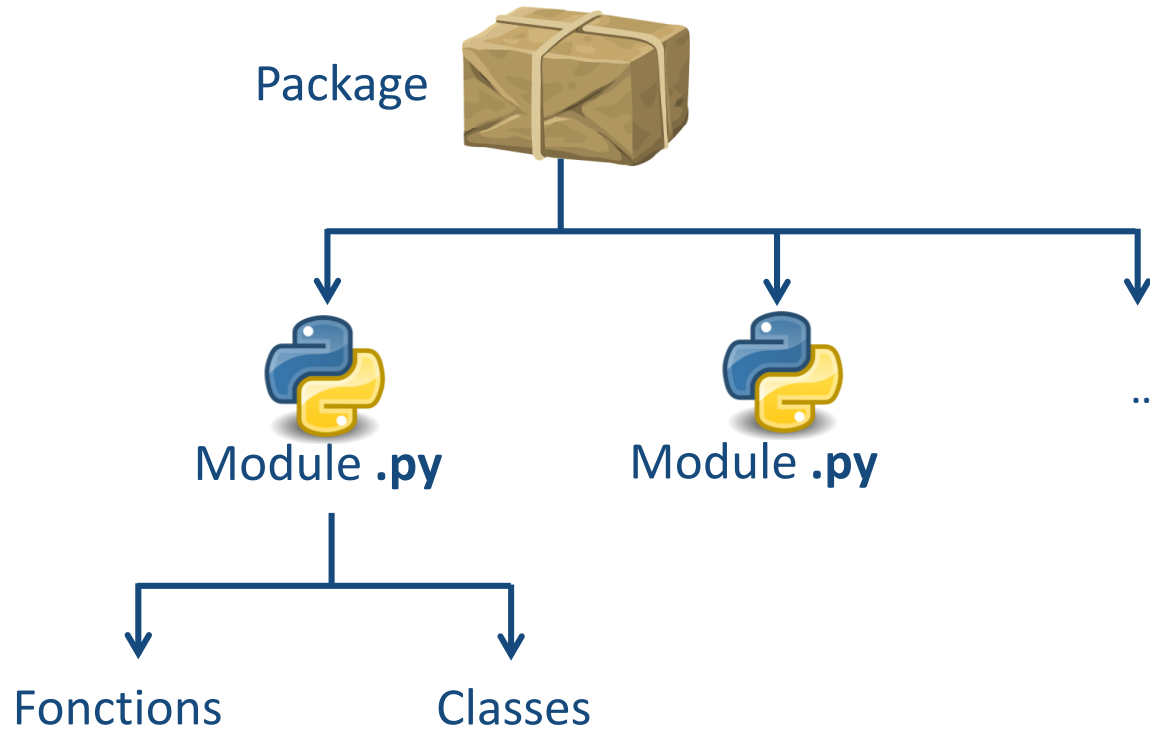
```
        _____
        print(i, end="")
else: print("AA")
```

- ☐ break
- ☒ continue
- ☐ print(i)

| MODULES



Modules



💡 Un module python est un fichier **.py** rassemblant des fonctions et des classes

Importation d'un module

- ☰ Pour accéder aux fonctions ou classes d'un module existant, il faut **charger le module** avec **import**

```
from nom_module import *
```

```
from nom_module import fct1, classe1, ...
```









```
import nom_module1 , nom_module2
```

```
import nom_module as nomLocal
```

```
from nom_package.nom_module import ...
```

```
import nom_package.nom_module
```

Quelques modules standards

-  **math** fournit des fonctions pour réaliser des calculs mathématiques complexes
-  **datetime** fournit des fonctions pour manipuler de façon simple ou plus complexe des dates et des heures ;
-  **json** permet l'encodage et le décodage de données au format JSON
-  **os** fournit des fonctions pour utiliser les fonctionnalités dépendantes du système d'exploitation
-  **re** fournit des opérations de *matching* des expressions régulières sur les chaînes de caractères
-  **random** fournit des fonctions pour simuler le hasard
-  **csv** implémente des classes pour lire et écrire des données tabulaires au format CSV
-  **sys** fournit un accès à certaines variables système utilisées et maintenues par l'interpréteur

⋮

Module **math**

Les fonctions mathématiques sont définies dans le module **math**

Pour importer **math** : **from math import ***

ou bien **import math**

factorial(x) : Retourne la factorielle de **x** (erreur si **x** n'est pas entier ou s'il est négatif)

exp(x) : Retourne e^x

log2(x) : Retourne le logarithme en base 2 de **x**

sqrt(x) : Retourne la racine carrée de **x**

ceil(x) : Retourne le plus petit entier, plus grand ou égal à **x**

⋮

<https://docs.python.org/fr/3.5/library/math.html>

Quiz

1. Pour charger le module math, il suffit de créer **from math import *** (une seule réponse)

- ☒ Vrai
- ☐ Faux

2. Quel module en Python supporte les expressions régulières ? (une seule réponse)

- ☒ re
- ☐ regex
- ☐ pyregex
- ☐ Aucune de ces réponses n'est correcte

3. Quelle est la sortie du code suivant ? (une seule réponse)

```
import math  
print(sqrt(10))
```

- ☐ 3.16
- ☐ 3
- ☒ Erreur

CHAPITRE 2

CONTENEURS STANDARDS EN PYTHON

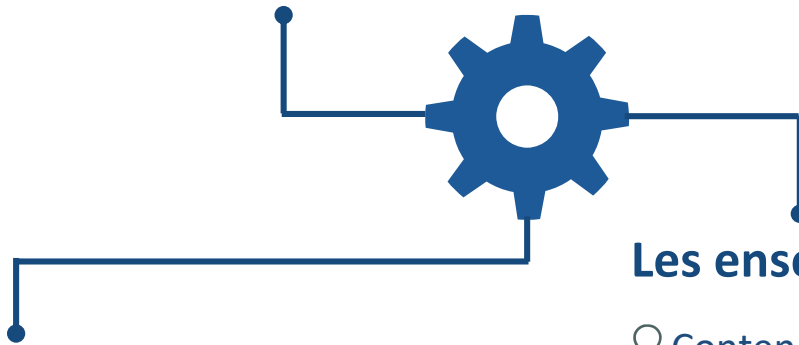


Conteneurs standard en Python

Les séquences

💡 Conteneur ordonné d'éléments indicés par des entiers

● Chaines de caractères (str), Listes (list) & Tuples (tuple)



Les dictionnaires (dict)

💡 Tableaux associatifs

💡 Stocker des couples « **clé-valeur** »

Les ensembles (set)

💡 Conteneur non ordonné d'éléments sans répétition

💡 Sans Indichage par des entiers



Mutable vs Immutable!

Tout variable en Python = un **objet**

Un **objet** → **Type + Id + value**

Tous les objets peuvent être soit **mutables**, soit **immutable**

💡 **Mutable** : Objets dont **la valeur peut être modifiée** sans changer d'Id

💡 **Immutable** : Objets dont **la valeur est inchangeable une fois qu'ils sont créés** sans changer d'Id

△ **Mutable** ☒ list ☒ set ☒ dict

△ **Immutable** ☒ bool ☒ int ☒ float ☒ tuple ☒ str

| CHAÎNE DE CARACTÈRES



Chaines de caractères

- Représentent toutes les informations qui ne sont pas numériques
- Apparaissent entre guillemets simples ' ', doubles " " et **triples** """ """

```
chaine1 = 'aaaa'  
chaine2 = "aaaaaaaa"  
chaine3 = """aaaaa bbbb cccccc  
dddd eeee pppppp"""  
  
print (chaine1) ; print(chaine2) ; print(chaine3)
```

← Textes sur plusieurs lignes

NB.

`print ('Aujourd'hui')` **SyntaxError: invalid syntax** ❌
`print ('Aujourd\'hui')` ou bien `print ("Aujourd'hui")`
↑
Insérer le caractère antislash « \ » v Utiliser « " » au lieu de « ' »

Accès & Slicing


`ch="Programmation Python"`

`print(ch[1]) # r` `print(ch[3]) # g`

`print(ch[-1]) # n` `print(ch[-20]) # P`

`print(ch[0:3]) # Pro` 0→2 `print(ch[2:5]) # ogr` 2→4

`print(ch[:3])` équivalent à `print(ch[0:3]) # Pro` 0→2

`print(ch[2:]) # ogrammation Python` 2→Fin

`print(ch[2::4]) # omiPo` 2→Fin (**pas=4**)

`print(ch[2::-1]) # orP` 2→Fin (**pas=-1**)

**Slicing ou découpage
en tranche**

[ind1:ind2] → ind1... ind2-1

Concaténation, répétition et modification

Concaténation et Répétition : + *

```
a="AA"+"BB"  
b="AB"*3  
print (a) # AABB  
print (b) # ABABAB
```

Les chaînes de caractères sont
immutable (non modifiable)

Modification d'un ou plusieurs caractère(s)

```
mot1 = "Programmation"  
mot2 = " Python"  
mot1=mot1+mot2  
print (mot1)
```







```
mot = "pytton"  
print (mot[3])  
mot[3]="h"  
print (mot[3])
```



NB.

Une fois une chaîne de caractères définie, vous ne pouvez plus modifier un de ses éléments, mais on peut définir une nouvelle chaîne avec le même identificateur

Fonctions élémentaires

-  `len(ch)` renvoie la longueur de `ch`
-  `ch.lower()` convertit une chaîne `ch` en minuscule
-  `ch.upper()` convertit une chaîne `ch` en majuscule
-  `ch.title()` convertit en majuscule l'initiale de chaque mot dans `ch`
-  `ch.capitalize()` convertit en majuscule seulement la première lettre de la chaîne `ch`
-  `ch.swapcase()` convertit toutes les majuscules dans `ch` en minuscules, et vice-versa

```
ch="Python Pour programmer"
print("Longueur de ch=",len(ch))
print(ch.lower())
print(ch.upper())
print(ch.capitalize())
print(ch.title())
print(ch.swapcase())
```

```
Longueur de ch= 22
python pour programmer
PYTHON POUR PROGRAMMER
Python pour programmer
Python Pour Programmer
pYTHON pOUR PROGRAMMER
```

Autres fonctions sur les chaines de caractères

- `ch.count(sch)` compte le nombre d'une sous-chaine `sch` dans `ch`
- `sch in/ not in ch` vérifie si une sous-chaine `sch` existe dans `ch`
- `ch.startswith(scr)` vérifie si une chaîne de caractères commence par une sous-chaine `sch`
- `ch.find(sch)` cherche la position d'une sous chaine `sch` dans `ch`
- `ch.index(sch)` retrouve l'indice de la première occurrence de la chaine `sch` dans `ch`

```
mot = "Hi python"
```

```
print(mot.count("on")) # 1
print(mot.count("n")) # 1
```

```
print ("HI" in mot) # False
print ("HI" not in mot) # True
print (mot.startswith("Hi")) # True
print (mot.startswith("Hi ")) # True
```

```
print(mot.find ("y")) # 4
print(mot.find ("tho")) # 5
print(mot.find ("a")) # -1
print(mot.find ("hoi")) # -1
```

```
print(mot.index("y")) # 4
print(mot.index("yl"))
```



```
"pypythonp".index("p",3) # 8
```

Autres fonctions sur les chaines de caractères

- `ch.split()` découpe `ch` en plusieurs éléments selon n'importe quelle combinaisons d'espaces blancs (' ', '\n', '\t')
- `ch.split(sep)` découpe `ch` en plusieurs éléments selon une chaîne séparateur `sep`
- `ch.replace(sch1, sch2)` remplace toutes les occurrences de `sch1` par `sch2`
- `ch.strip()` permet de nettoyer les bords de `ch`
- `ch.isdigit()` vérifie si `ch` se compose uniquement de chiffres

```
ch="AA dd; bb \n c; K"
print(ch.split()) #['AA', 'dd;', 'bb', 'c;', 'K']
print(ch.split(";")) #['AA dd', ' bb \n c', ' K']
print(ch.split(";", maxsplit=1))
#['AA dd', ' bb \n c; K']
```

Le nombre de fois qu'on souhaite découper la chaîne

```
mot = "pytton"
mot=mot.replace("t", "h")
print (mot) # pyhhon
```

```
mot = "pytton"
mot=mot.replace("tt", "th")
print (mot) # python
```

```
ch = "AABfs2234BBDDS"
print (ch.isdigit()) # False
```


| LISTES & TUPLES



Listes

Représentent une série de valeurs (objets) qui peuvent être de types différents

NB. Une liste est déclarée par une série de valeurs, séparée par des virgules, et le tout encadré par des crochets []

```
l1=["ABC",20,34,"M"]
```

```
print(l) # ['ABC', 20, 34, 'M']
```

```
l2=[] équivalent à l2=list()
```

```
print(l2) # []
```

Une liste vide se définit par [] ou list()

```
l3=[20,30,5]
```

```
for i in l3:
```

```
    print(i) # ?
```

Ou bien

```
for i in _____ ? :
```

```
    print(l3[i])
```

← Parcours d'une liste

Accès et Slicing

0 1 2 3 4 → **Indiçage positif**
l = [11, 'abc', 'ddd', 4, 12]
← -5 -4 -3 -2 -1 **Indiçage négatif**

print(l[1]) # abc print(l[-1]) # 12

print(l[:]) équivalent à print(l) # [11, 'abc', 'ddd', 4, 12]

print(l[2:]) # ['ddd', 4, 12] (à partir de l'élément d'indice 2 vers la fin)

print(l[:3]) # ['ddd', 4, 12] (à partir de la 1^{er} élément à celui d'indice 3-1=2)

print(l[-1:-3]) # [] (l[val1:val2]) val1 < val2

print(l[:-3]) # [11, 'abc']
0 → -3-1 = -4

print(l[-3:-1]) # ['ddd', 4]
-3 → -1-1 = -2

print(l[::2]) # [11, 'ddd', 12] (toutes les éléments de l par pas de 2)

print(l[1:3:2]) # ?

Modification des éléments d'une liste

Les listes sont **mutables** : modifiables par l'ajout, le remplacement ou la suppression d'un objet

```
l=[20,29,13,17]
```

```
l[1]=5 ; print(l) # [20,5,13,17]
```

```
l[1:3]=[4,8] ; print(l) # [20, 4, 8, 17]
```

```
l[1:3]=[1,2,3] ; print(l) # [20, 1, 2, 3, 17]
```

```
l[-1]=4 ; print(l) # [20, 1, 2, 3, 4]
```

```
l[:2]=[0,-2,-4] ; print(l) # [0, 1, -2, 3, -4]
```

```
l[:2]=[0,-2,-4,-6] ; print(l) # ☒
```

```
l1=[0,1,2]
```

```
l2=["AA",2,5]
```

```
print(id(l2)) # 1828080535560
```

```
l2=l1+l2 ← Concaténation
```

```
print(l2) # [0,1,2,"AA",2,5]
```

```
print(id(l2)) # 1828080535560
```







```
l2=l1*2 ← Répétition
```

```
print(l2) # [0,1,2,0,1,2]
```

```
l2=[l1,l2] ← Liste de listes
```

```
print(l2) # [[0,1,2],["AA",2,5]]
```

Opérations sur les listes





-  `len(lis)` renvoie la longueur de la liste `lis`
-  `min(lis)` renvoie l'élément minimum dans la liste `lis`
-  `max(lis)` renvoie l'élément maximum dans la liste `lis`
-  `sum(lis)` renvoie la somme des éléments de la liste `lis`
-  `lis.count(elem)` compte le nombre de l'élément `elem` dans la liste `lis`
-  `list` & `range` : génère une liste d'entiers

```
l1=["AB",20,6,1]
print(len(l1)) # 4
print(max(l1)) ❌

l2=[11,46,1,1]
print(max(l2)) # 46
print(sum(l2)) # 59
print(l2.count(1)) # 2

l3=list(range(1,4))
print(l3) # [1, 2, 3]
```

Opérations sur les listes

-  `el in lis`, `el not in lis` : vérifie l'appartenance d'un élément `el` dans `lis`
-  `lis.append(elem)` ajoute l'élément `elem` à la fin de la liste `lis`
-  `lis.insert(ind, elem)` insère l'élément `elem` dans l'indice `ind` de `lis`
-  `lis.extend(seq)` ajoute le contenu d'une sequence `seq` à `lis`
-  `lis.sort()` trie la list `lis` Les éléments de `lis` doivent être de type str
-  `sep.join(lis)` convertie la liste `lis` en une chaine de caractères en ajoutant un séparateur `sep` entre chaque élément

```
l=["AA",20,31]
print("AA" in l)    # True
print(13 not in l)  # True

l.append(20)
print(l)    # ['AA', 20, 31, 20]
l.insert(2,"kk")
print(l)    # ['AA', 20, 'kk', 31, 20]
l.extend([1,2]) # ['AA', 20, 'kk', 31, 20, 1, 2]
l.sort() ⓧ
l1=["BB","DD","AA"]
l1.sort()
print(l1) # ['AA', 'BB', 'DD']
l1.sort(reverse=True) ← Ordre décroissant
print(l1) # ['DD', 'BB', 'AA']

ch="-".join(l1)
print(ch) # DD-BB-AA
```

Opérations sur les listes

- `lis.remove(elem)` supprime l'élément `elem` de la liste `lis` (s'il existe)
- `del lis`
- `lis.pop()` supprime et retourne le dernier élément de `lis`
- `lis.pop(i)` supprime et retourne l'élément à la position `i` de `lis`
- `lis.reverse()` inverse l'ordre des éléments de la liste `lis`
- `reversed (lis)` affiche la liste inversé de `lis` sans l'affecter

```
l=[14, 7, 12.1, 7, "KK","DD"]
l.remove(3) ❌
l.remove(7)
print(l) # [14, 12.1, 7, 'KK', 'DD']

val1=l.pop()
val2=l.pop(2)
print(val1, val2, l)
# DD 7 [14, 12.1, 'KK']
l.reverse()
print(l) # ['KK', 12.1, 14]

reversed(l)
print(l) # ['KK', 12.1, 14]
for i in reversed(l):
    print(i, end=" ")# 14 12.1 KK

del l # ?
del l[:] # ?
```

Listes en compréhension

- Listes dont le contenu est défini par l'application d'une fonction à une séquence d'éléments (filtrage)

```
liste = [<expression> for <element> in <iterable> if <condition> ]
```

Facultatif

```
liste = [2*i for i in range (0,6)]
print(liste)    # [0, 2, 4, 6, 8, 10]
liste = [i for i in range (0,10) if i%2!=0]
print(liste)    # [1, 3, 5, 7, 9]
liste = [0 if i%2!=0 else 1 for i in range (0,10) if i>0 ]
print(liste)    # [0, 1, 0, 1, 0, 1, 0, 1, 0]
liste = [[0 if i!=j else 1 for i in range (0,3)] for j in range (0,3)]
print(liste)    # [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```


Module **random**

Listes & Valeurs aléatoires

`from random import *`

`randint(a, b)`=`randrange(a,b)` génère un nombre entier aléatoirement dans l'intervalle `[a,b]`

`randrange(a, b, p)` génère un nombre entier aléatoirement dans l'intervalle `[a,b]` par un pas `p`

`sample(seq, n)` génère une liste de `n` valeurs aléatoires sans répétition, inclus dans une séquence `seq`

`shuffle(lis)` mélange aléatoirement `lis`

```
from random import *
```

```
L1 = [randrange(0,10,2) for i in range(5)]
```

```
print("L1=",L1) # L1= [6, 2, 0, 8, 4]
```

```
L2 = [randint(0,10) for i in range(5)]
```

```
print("L2=",L2) # L2= [6, 2, 0, 8, 4]
```

```
L3 = sample(range(1,30), 5)
```

```
L4 = sample([30,46,33,21,78,34,35], 5)
```

```
print("L3=",L3) # L3= [8, 2, 21, 23, 26]
```

```
print("L4=",L4) # L4= [33, 21, 35, 46, 34]
```

```
L5 = ['aaa',4,'bbbb',8,10]
```

```
shuffle(L5) ; print(L5)
```

```
# ['aaa', 8, 10, 'bbbb', 4]
```

Tuples

Utilisent des **()** au lieu des **[]**, elles sont facultatives mais recommandées

Peuvent contenir des objets de types différents

t1=() #ou bien **t1=tuple()** ← Un Tuple vide se définit par **()** ou **tuple()**

t1=(20, "ABC", 3.2) # ou bien **t1= 20, "ABC", 3.2**

print(t1) # **(20, 'ABC', 3.2)**

t1=20, ← Un Tuple d'un seul élément (**virgule obligatoire**)

t2=(42,12,55) ; t2[0]=10 **⊗** ← Les éléments d'un Tuple sont **immutables**

t3=(2,14) ; print(id(t3)) # **1828081314184**

t3=t2+t3 **print(t3)** # **(42, 12, 55, 2, 14)** ← concaténation

print(id(t3)) # **1828062559400**

t3=(t2,t3) # ou bien **t3=t2,t3** ← Tuple de Tuples

print(t3) # **((42, 12, 55), (2, 14))**

Empaquetage & Dépaquetage

L'utilisation de la virgule **lors de la création** d'un Tuple, permettant de séparer les éléments les uns des autres, s'appelle **empaquetage automatique**

`t=2, 5, 7`; print(t) ← **Empaquetage automatique**

L'utilisation de la virgule **dans la partie gauche** d'une assignation dont l'expression droite retourne un Tuple s'appelle **dépaquetage automatique**

`a, b, c=t` ← **Dépaquetage automatique**

print("a=", a) # **a= 2**

print("b=", b) # **b= 5**

print("c=", c) # **c= 7**

*var & _

Le symbole `_` peut être utilisé comme un nom de **variable jetable** si l'on a seulement besoin de quelques éléments d'un Tuple

Un préfixe `*var` convertit `var` en une liste

```
a=1,2,3,4 ; x,_,y,_=a
print(x,y) # 1 3
```

b est une variable de type list

```
a,*b,c=1,2,3,4
print(a,b,c, sep=" | ") # 1 | [2, 3] | 4
```

```
*b,c=1,2,3,4
print(b,c, sep=" | ") # [1, 2, 3] | 4
```

`a,b=1,2,3,4 ; print(a, b)` ❌ Too many values to unpack (expected 2)

`*b=1,2,3,4 ; print(b)` ❌ Starred assignment target must be in a list or tuple

Opérations sur les Tuples

- `len(tup)` calcule la longueur du Tuple `tup`
- `min(tup)` retourne l'élément le plus petit dans `tup`
- `max(tup)` retourne l'élément le plus grand dans `tup`
- `tup.count(elem)` retourne le nombre d'occurrences de l'élément `elem` dans `tup`
- `tup.index(elem)` : retourne la première occurrence de l'élément `elem` s'il existe dans `tup`, **Erreur** sinon
- `sep.join(tup)` convertie le Tuple de chaînes de caractères `lis` en une chaîne de caractères en ajoutant un séparateur `sep` entre chaque élément

```
t=(11,29,"EDz",19.65,"EDz")
t1=(5,10)
print(len(t)) # 5
print(max(t1)) # 10
print(max(t)) ❌
print(t.count("aaa")) # 0
print(t.count("EDz")) # 2
```

```
print(t.index("E")) ❌
```

```
t3=("DDD","FFF")
ch="--".join(t3) ; print(ch)
# DDD--FFF
```