

[{*m. Aly*}]

# Python Niveau 1

Aly Tall NIANG

# Hello World!

- Ouvrir le Terminal
- Taper la commande `python[version]` puis valider pour entrer dans un **interpréteur python**

## Exercice:

afficher le message ***Hello World***

- Pour afficher un message, on utilise la fonction `print("Message à afficher")`

**Remarque:** Quand on est dans l'**interpréteur Python**, on a pas besoin d'utiliser ***print()*** pour afficher un message ou un résultat car la ligne de code sera directement interprétée et le résultat affiché.

```
>>> print("Hello World")  
Hello World
```

```
>>> "Hello World"  
'Hello World'
```

```
>>> Hello World  
File "<stdin>", line 1  
Hello World  
^^^^^
```

```
SyntaxError: invalid syntax
```

```
>>> print(3+7)  
10
```

```
>>> 3+7  
10
```

```
>>>
```

# Les types natifs

# Les types natifs

Les types natifs sont des types de données préexistants qui nous permettent de représenter les données à manipuler.

Il existe plusieurs types natifs tel que:

- les chaînes de caractères
- les nombres
- les booléens

**NB** : il existe d'autres types natifs tels que les *types séquentiels*, les *types d'ensembles* ou les *types de correspondances* qui sont des façons d'organiser et d'agencer ces trois premiers types.

# Les chaînes de caractères (str ou string)




- une chaîne de caractères est un ensemble de caractères communément appelée du **texte**.
- en programmation on utilise souvent le mot **chaîne** plutôt que l'expression **chaîne de caractères**.
- une chaîne peut ne pas contenir de caractère : **une chaîne vide** (représentée généralement par une paire de guillemet vide).
- une chaîne peut contenir autant de caractères que l'on souhaite.
- En python les caractères d'une chaîne (0 à plusieurs) sont entourés par des guillemets (simple: '...' ou double : "...")
- **Exemples :**
  - "Here is an example of string", "01234566789", "3", ""
  - 'Here is an example of string', '01234566789', '3', "

# Les chaînes de caractères

## Remarque :

l'utilisation des guillemets simples ou doubles depend entierement de vous, néanmoins, si parmi les caractères de la chaîne, il y a au moins un guillemet simple ou double (apostrophe par exemple) alors il faudra l'échapper (avec le caractère **antislash** : \) ou changer les guillemets qui entoure la chaîne.

## Exemple :

- "Let's go!" 
- 'Let's go!' 
- 'Let\'s go!' 

# Les chaînes de caractères

- Les chaînes de caractères **multilignes** :
  - Le nombre de caractères à représenter peut être très grand au point qu'on veuille représenter la chaîne sur **plusieurs lignes**.

On utilise alors les triples guillemets pour entourer les caractères : `"""` ou `'''`.

Dans ce cas, les guillemets à l'intérieur de la chaîne ne poseront pas de problèmes.

## Exemples :

```
[>>> print("""Let's go! Here is an example of string
... written on several lines""")
Let's go! Here is an example of string
written on several lines
>>> █
```



# Les chaînes de caractères

- Les caractères spéciaux:

Il existe pour certains caractères, une interprétation particulière lorsqu'ils sont précédés d'un antislash.

Exemples:

```
[>>> print("a first line\u000a second line")
```

```
a first line
```

```
a second line
```

```
[>>> print("I \u2764 Africa")
```

```
I ♥ Africa
```

```
>>>
```

# Les chaînes de caractères

Remarque :

- Pour éviter que certains caractères ne soient interprétés par Python, on peut utiliser les chaînes de caractères brutes (Raw String).
- Il suffit de mettre un “**r**” devant la chaîne.
- Exemples :
  - Afficher un chemin de fichier sur windows : `c:\programs\nouveau\example.py`
  - Afficher le message suivant : `le code du symbole coeur est : \u2764`

# Les chaînes de caractères

```
[>>>  
[>>> print(r"c:\programs\nouveau\exemple.py")  
c:\programs\nouveau\exemple.py  
[>>> print(r"le code caractere du coeur est: \u2764")  
le code caractere du coeur est: \u2764  
>>> █
```

# Les chaînes de caractères

Exemples de caractères interprétés par Python:

- `\a`    => bip sonore
- `\b`    => retour en arrière
- `\f`    => saut de page
- `\n`    => retour à la ligne
- `\r`    => retour chariot
- `\t`    => tabulation horizontale
- `\v`    => tabulation verticale

# Les nombres

Il existe en python deux types de nombres:

- **Les nombres entiers (int):**

Les nombres qui n'ont pas de partie décimale.

Exemples : 123 / 0 / -665 / 154645372 / -1 / 1\_000\_000 (python 3.6 ou plus)

- **Les nombres décimaux (float):**

Les nombres qui ont une virgule ( . ), on les nomme aussi nombre flottant.

Exemples: -4.89 / 90.0 / 1093.22 / 1\_000.123\_456 (python 3.6 ou plus)

# Les booléens (bool)

Un booléen en python est un objet qui ne peut prendre que valeur possible :

- **True** (Vrai)
- **False** (Faux)

Les booleen sont en réalité une sous classe des nombre entier puisque **True** est l'équivalent de **1** et **False**, l'équivalent de **0**.

`issubclass(bool,int)` renvoie True pour confirmer nos dires.

Remarque : on peut additionner des booléens avec des nombres :

- `29 + True` ==> 30
- `25 + False` ==> 25

# Les booléens (bool)

En programmation, tous les objets (variables) peuvent avoir l'état **Vrai** ou **Faux**.

On peut utiliser la fonction `bool()` pour tester si un objet est Vrai ou Faux.

Exemples :

- `bool("Hello world")`      => True
- `bool("")`      => False

Tous les Objets ont par défaut une valeur qui est considérée comme Faux et les autres valeurs comme Vrai :

`" "`      => False

`0`      => False

`0.0`      => False

`[]`      => False

`{}`      => False

# Les types natifs (fin)

Avec Python, on peut créer des objets des types natifs à partir des **constructeurs** (fonction permettant de créer l'objet) de la classe du type correspondant.

## Exemples :

**int(12)**

=> 12

**float(1.65)**

=> 1.65

**str("Hello")**

=> "Hello"

**bool(True)**

=> True

**str(12)**

=> "12"

**int("23")**

=> 23

## Remarque :

-> On peut donc utiliser ces fonctions pour convertir une valeur d'un type vers un autre.

-> On peut utiliser la fonction **type()** pour connaître le type d'un objet. Ex: **type('toto')** => str



# Les variables

# Les variables

- Les variables nous permettent de stocker des valeurs (données) tout au long de l'exécution du programme.
- Ainsi la valeur stockée dans la variable pourra être modifiée ou lue à partir du nom de la variable.
- Déclaration d'une variable:
  - **nom\_variable = valeur**

Python est assez intelligent pour savoir le type de la variable en se basant sur sa valeur. Le type de la variable dépend donc de sa valeur.

Exemples:

**x = 10**

**name= "Toto"**

**test = True**

**age = int(10)**

**lang = str("Python")**

**NB** : on peut utiliser la fonction **id()** pour connaître l'emplacement d'une variable dans la RAM.

# Les variables : Affectations

Affecter une valeur à une variable revient à lui assigner une valeur dans le programme.

Il existe plusieurs façon d'affectation en Python parmi lesquelles :

- **Affectations simples**
  - `nom_var1 = valeur1`
  - `nom_var2 = valeur2`
- **Affectations parallèles**
  - `nom_var1, nom_var2 = valeur1, valeur2`
- **Affectations multiples**
  - `nom_var1 = nom_var2 = valeur_commune`

# Les variables : Règles de nomenclature

Voici des règles à respecter pour nommer une variable :

- Deux variables ne peuvent pas porter le même nom (on parle d'identificateur).
- Ne peut pas commencer par un chiffre
- Ne peut pas contenir d'espace
- Ne peut contenir que des caractères alphanumériques (lettres, chiffre, le tiret du bas : \_)
- Ne pas être un **mot réservé** à Python (print, True, type,...)
- Exemples:

## Erreur

- True
- #my tag
- Ma-Variable
- 221SN
- nom école

## Valide

true  
myTag  
Ma\_Variable  
SN221  
nom\_école (Python 3 +)

## Recommandé

true  
my\_tag  
ma\_variable  
sn\_221  
nom\_ecole

# La fonction `input()`

Il y a deux façon de mettre une valeur dans une variable:

- **Affectation** (par le programmeur):
  - `x = 12`
  - `nom = "Toto"`
- **Saisie** (par l'utilisateur) :
  - On utilise la fonction `input` qui renvoie une chaîne.
  - Exemple :

```
■ name = input("Enter your name:")
```

```
■ age = input("Enter your age")
```

*Quelque soit la valeur saisie, la fonction **`input()`** renvoie toujours une chaîne.*

```
[>>> name=input("Enter your name : ")
[Enter your name : toto
[>>> print(name)
toto
[>>> type(name)
<class 'str'>
[>>> age=input("Enter your age : ")
[Enter your age : 16
[>>> print(age)
16
[>>> type(age)
<class 'str'>
[>>>
```

# Série d'exercices



# Manipulation de chaînes de caractères



# Manipulation de chaînes de caractères

Les chaînes de caractères sont un peu complexes à manipuler. Pour cela, Python met à notre disposition des fonctions prédéfinies de manipulation de chaînes parmi lesquelles :

- **Fonctions de modification de la casse:**

- **upper()** : Elle met tous les caractères d'une chaîne majuscule.
  - `"Hello World".upper()`                      =>    `"HELLO WORD"`
- **lower()** : Elle met tous les caractères d'une chaîne minuscule.
  - `"Hello World".lower()`                      =>    `"hello world"`
- **capitalize()** : Elle met le premier caractère de la chaîne en majuscule.
  - `"python is awesome!".capitalize()`                      =>    `"Python is awesome1"`
- **title()** : Elle met le premier caractère de chaque mot en majuscule.
  - `"python is awesome!".title()`                      =>    `"Python Is Awesome!"`

# Manipulation de chaînes de caractères

- Fonctions de remplacement de chaînes :

- **replace(ch1,ch2)** : Elle remplace chaque occurrence de la chaîne **ch1** par la chaîne **ch2**
  - `"Good night".replace("night", "morning")` => `"Good morning"`
  - `"Py-then".replace("-", "").replace("e", "o")` => `"Python"`
- **strip()** : elle enlève les espace de début et de fin de chaîne.
  - `" Good morning ".strip()` => `"Good morning"`
- **strip(ch)** : elle supprime chaque caractère d'une chaîne en partant du début puis de la fin si le caractère est présent dans la chaîne **ch**. La suppression s'arrête au premier caractère qui n'est pas présent dans la chaîne **ch**.
  - `"question".strip("inoqu")` => `"est"`
- **rstrip()** & **lstrip()** : la recherche se fait à droite uniquement (r) ou à gauche (l).

# Manipulation de chaînes de caractères

- Fonctions de séparation et de jonction :

- `split()` :

- Elle permet de séparer des parties d'une chaîne puis met chaque partie dans une cellule d'une **liste**.

- `"1,2,3,4,5".split(",")`  $\Rightarrow$  `['1', '2', '3', '4', '5']`

- `join()` :

- Elle permet de joindre une liste de caractères en chaîne, en précisant le(s) caractère(s) de jointure.

- `"-".join(['1','2','3'])`  $\Rightarrow$  `"1-2-3"`

- Fonction de remplissage :

- `zfill()` :

- Elle permet de remplir des zéro (0) devant une chaîne donnée.

- `"5".zfill(4)`  $\Rightarrow$  `"0004"`

- `"1000".zfill(4)`  $\Rightarrow$  `"1000"`

# Manipulation de chaînes de caractères

- Les fonctions de test `is_` :

Il existe plusieurs fonctions commençant par **is** et qui renvoient **True** ou **False**. Elles permettent de tester le format d'une chaîne de caractères.

- Exemples:

- `isdigit()` : qui renvoie **True** si la chaîne testée ne contient que des chiffres.
  - `"123947".isdigit()`                      =>    **True**
  - `"3.14".isdigit()`                        =>    **False**
- `istitle()`
- `isupper()`
- `islower()`
- ...

# Manipulation de chaînes de caractères

- **Fonction pour compter le nombre d'occurrences :**

- **count()** : Elle renvoie le nombre d'occurrence d'une chaîne.

- `"bonjour tout le monde".count("o")`      =>      **4**

- **Fonctions de recherche de position de chaîne:**

Elles permettent de rechercher la position de la première occurrence d'une chaîne dans une chaîne principale. La première position est zéro (0).

- **find()** :

- `"Hello world".find("o")`      =>      **4**

- `"Hello world".find("word")`      =>      **-1**

- Il existe la fonction **rfind()** qui recherche à partir de la fin.

- **index()** :

- `"Hello world".index("o")`      =>      **4**

- `"Hello world".find("word")`      =>      **Erreur**

# Manipulation de chaînes de caractères

- Fonctions de recherche en début ou fin de chaîne :

- **startswith():**

- Elle teste si une chaîne commence par une autre chaîne donnée.

- `"77 000 00 00".startswith("76")`                      =>    `False`

- **endswith() :**

- Elle teste si une chaîne se termine par une autre chaîne donnée.

- `"toto@exemple.com".endswith(".com")`                      =>    `True`

# Série d'exercices

# Les opérateurs



# Les opérateurs

- Les opérateurs mathématiques :

```
>>> print(1+2)
3
>>> print(1-2)
-1
>>> print(2*2)
4
>>> print(2/2)
1.0
>>> print("to"+"to")
toto
>>> print("to"*3)
tototo
>>> █
```

# Les opérateurs

- **Autres opérateurs mathématiques :**

- **% :**
  - Modulo, permet de récupérer le reste d'une division.
  - `10 % 3`  $\Rightarrow$  `1`
  - `10 % 5`  $\Rightarrow$  `0`
- **// :**
  - Division entière.
  - `10 / 3`  $\Rightarrow$  `3.3333333`
  - `10 // 3`  $\Rightarrow$  `3`
- **\*\* :**
  - Puissance
  - `2 * 4`  $\Rightarrow$  `8`
  - `2 ** 4`  $\Rightarrow$  `16`

# Les opérateurs

- **Opérateur d'affectation :**

- **= :**
  - Il permet de donner une valeur à un objet (variable).
  - `name = "toto"`      `x=200`

- **Opérateurs d'accumulation :**

- Ils permettent d'assigner une nouvelle valeur à une variable en se basant sur son ancienne valeur.
- |            |          |               |          |              |
|------------|----------|---------------|----------|--------------|
| <b>+=</b>  | <b>:</b> | <b>x=x+3</b>  | <b>⇔</b> | <b>x+=3</b>  |
| <b>-=</b>  | <b>:</b> | <b>y=y-2</b>  | <b>⇔</b> | <b>y-=2</b>  |
| <b>*=</b>  | <b>:</b> | <b>z=z*3</b>  | <b>⇔</b> | <b>z*=3</b>  |
| <b>/=</b>  | <b>:</b> | <b>a=a/4</b>  | <b>⇔</b> | <b>a/=4</b>  |
| <b>//=</b> | <b>:</b> | <b>b=b//5</b> | <b>⇔</b> | <b>b//=5</b> |
| <b>%=</b>  | <b>:</b> | <b>c=c%2</b>  | <b>⇔</b> | <b>c%=2</b>  |
| <b>**=</b> | <b>:</b> | <b>d=d**4</b> | <b>⇔</b> | <b>d**=4</b> |

# Les opérateurs

- **Opérateurs de comparaison ou relationnels :**

Ils sont utilisés avec des structures conditionnelles et le résultat est True ou False.

- **>** : supérieur à
- **<** : inférieur à
- **>=** : supérieur ou égale à
- **<=** : inférieur ou égale à
- **==** : égale à
- **!=** : différent
- **Remarque avec l'opérateur is :** l'opérateur **is** permet de tester si deux objets(variable) sont les mêmes (même adresse mémoire), il est différent de l'opérateur égale(==) qui teste l'égalité (si deux objets ont les mêmes valeurs). *Les nombres de -5 à 256 ont tous la même adresse.*

# Les opérateurs

- Opérateurs logiques :

Ils sont aussi utilisés avec des structures conditionnelles. Ils permettent de combiner plusieurs conditions en même temps ou bien de prendre le contraire d'un résultat.

- or** : OU      `True or False => True`      `True or True => True`      `False or False => False`
  - and** : ET      `True and False => False`      `True and True => True`      `False and False => False`
  - not** : NON      `not(True) => False`      `not(False) => True`
- 
- NB** : Il reste d'autres opérateurs en python tels que les opérateurs binaires ou spéciaux qu'on va explorer au fur et mesure qu'on avance dans le cours.

# Série d'exercices



# Formatage de chaînes de caractères

# Formatage de chaînes de caractères

- **Concaténation** : Pour coller des chaînes de caractères entre elles.
  - `print("Toto" + " " + "et" + " " + "Titi")`  $\Rightarrow$  `'Toto et Titi'`
  - `print("Toto" + 10)`  $\Rightarrow$  **Erreur**
  - `print("Toto" + str(10))`  $\Rightarrow$  `'Toto10'`
- **les f-string** : une manière beaucoup plus élégante de concaténer des chaînes et des nombres sans pour autant les convertir ou d'intégrer du code python à l'intérieur d'une chaîne de caractères (**Python 3.6+**).

Exemples :



```
[>>> name = "Toto"  
[>>> age = 14  
[>>> print(f"nom : {name} et age : {age} ans")  
nom : Toto et age : 14 ans
```

```
[>>> x = 3  
[>>> y = 2  
[>>> print(f"{x} à la puissance {y} est {x ** y}")  
3 à la puissance 2 est 9
```

```
[>>>
```

# Formatage de chaînes de caractères

- **format()** : Pour introduire des valeurs dans une chaîne à des emplacements déterminés : {}.
  - On peut utiliser :
    - des paires d'accolades vide
    - des paires d'accolades avec des nom
    - des paires d'accolades avec des indices

## Exemples:

[ $\{m.\text{aly}\}$ ]

```
[>>> name="Toto"
>>> age=14
>>>
>>> print("Je m'appelle {} et j'ai {} ans".format(name,age))
Je m'appelle Toto et j'ai 14 ans
>>>
>>> print("Je m'appelle {n} et j'ai {a} ans".format(n=name,a=age))
Je m'appelle Toto et j'ai 14 ans
>>>
>>> print("Je m'appelle {n} et j'ai {a} ans".format(n="Titi",a=16))
Je m'appelle Titi et j'ai 16 ans
>>>
>>> print("Je m'appelle {0} et j'ai {1} ans".format(name,age))
Je m'appelle Toto et j'ai 14 ans
>>>
>>> print("Je m'appelle {1} et j'ai {0} ans".format(name,age))
Je m'appelle 14 et j'ai Toto ans
>>>
>>> print("J'ai {0} et je suis le numero {0}".format(age))
J'ai 14 et je suis le numero 14
>>>
```

# Série d'exercices

# Les Structures conditionnelles

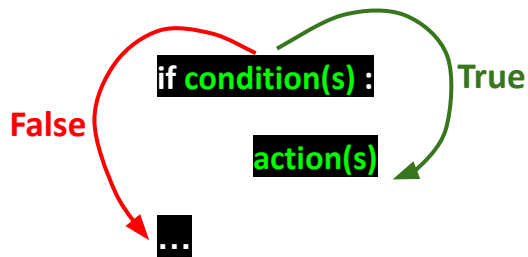
# Les structures conditionnelles

- Elle nous permet de tester une ou plusieurs conditions pour pouvoir effectuer une ou plusieurs actions en fonction du résultat de ce test.
- une structure conditionnelle est une question **directe**, on répond par **oui (True)** ou **non (False)**.
- Plusieurs conditions peuvent être combinées, dans ce cas on suit les règles suivantes pour connaître le résultat final:
  - Si une condition générale est composée de plusieurs sous-conditions séparées par des **OR** alors la condition générale est vérifiée si **au moins** une des sous-conditions est **Vraie**.
  - Si une commission générale est composée de plusieurs sous-conditions séparées par des **AND** alors la condition générale est vérifiée si **toutes** les sous-conditions sont vérifiées aussi.

# Les structures conditionnelles

- **condition simple :**

- Cette structure indique **uniquement** le(s) action(s) à faire si une ou plusieurs conditions **sont vérifiées**.
- **syntaxe :**




**Remarque :** ne pas oublier les **deux-points** et l'**indentation** du code : on parle de bloc d'instructions.

# Les structures conditionnelles

- condition simple (Exemple) :

```
1 age = 19
2
3 if age >= 18:
4     print("Your are of legal age!")
5
```





# Les structures conditionnelles

- **Bloc d'instruction :**

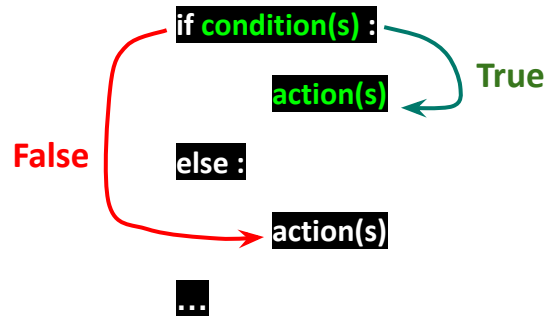
- Un bloc d'instructions est un ensemble de lignes de code qui dépend d'une autre ligne.
- Avec les autres langages de programmation, les blocs sont bien entourés par des accolades.
- Avec **python** c'est le **formatage** du document qui indique les différents blocs.
- Toutes les instructions d'un **même bloc** doivent être au **même niveau d'indentation**.
- Le début d'un bloc est signalé par les **deux-points** de fin de ligne ( : )
- On peut imbriquer plusieurs blocs d'instructions les un dans les autres mais faire **attention à l'indentation**.
- indenter le code avec la touche tabulation ou en appuyant 4 fois sur la touche espace.
- **Exemple :**

```
1 note = 19
2
3 if note >= 10:
4     print("Génial vous avez la moyenne!\n")
5     if note >= 16:
6         print("Félicitation.\n")
7     print("Au revoir!")
8
```

# Les structures conditionnelles

- **conditions alternatives :**

- Cette structure indique en même temps le(s) action(s) à faire si une ou plusieurs conditions sont **vérifiées** mais aussi le(s) actions action(s) à faire dans le cas **contraire**. On aura donc **deux blocs d'instructions** qui seront jamais exécutés en **même temps**.
- syntaxe :



# Les structures conditionnelles

- conditions alternatives (Exemple) :

```
1 note = 19
2
3 if note >= 10:
4     print("Génial vous avez la moyenne!")
5 else:
6     print("Désolé, vous n'avez pas la moyenne!")
7
```

# Les structures conditionnelles

- plusieurs conditions (Exemple) :

```
1 note = 19
2
3 if note >= 10:
4     print("Désolé, vous n'avez pas la moyenne!")
5 else:
6     if note > 16:
7         print("Félicitation vous avez une bonne moyenne!")
8     else:
9         print("Génial Vous avez la moyenne!")
10
```

# Les structures conditionnelles

- plusieurs conditions avec **elif** (Exemple) :

```
1 note = 19
2
3 if note >= 10:
4     print("Désolé, vous n'avez pas la moyenne!")
5 elif note > 16:
6     print("Félicitation vous avez un bonne moyenne!")
7 else:
8     print("Génial Vous avez la moyenne!")
9
```

# Les structures conditionnelles

- **Opérateur ternaire :**

- C'est une structure conditionnelle particulière que l'on peut écrire sur une même ligne. Elle renvoie une valeur si la condition est vérifiée et une autre valeur sinon. la valeur renvoyée est en general stockée dans un objet (variable).
- **syntaxe:**

```
variable = valeur_si_vrai if condition(s) else valeur_si_faux
```

**Exemple:**

# Les structures conditionnelles

```
1 age = 19
2
3 majeur = True if age >= 18 else False
4
```



# Série d'exercices

# Les structures itératives

# Structure itératives

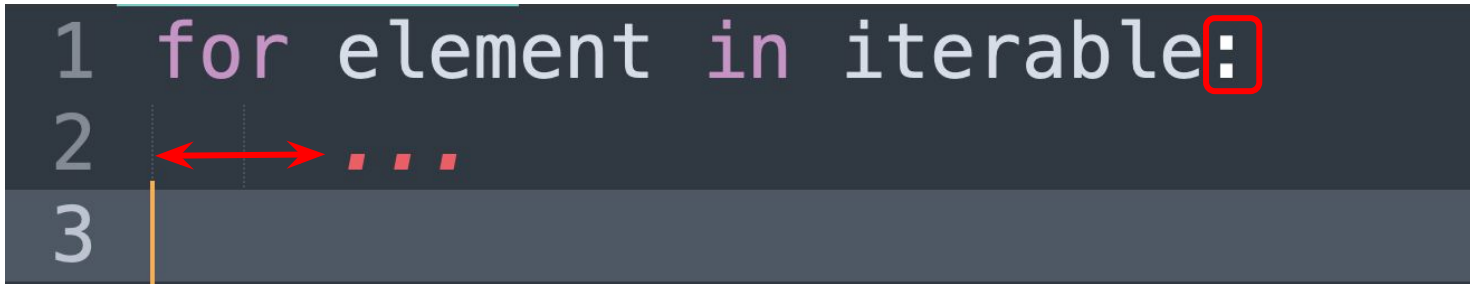
- Les structures itératives font partie des fondements de la programmation.
- Elles nous permettent de ne pas se répéter pour certaines instructions
- Elles servent aussi à parcourir des structures de données,
- Elles nous permettent de répéter une opération un certain nombre de fois
- Elles sont aussi utilisées pour contrôler la saisie,
- ...
- Il existe deux types de boucle en python : la boucle **for** et les boucles **while**

# Structures itératives : Boucle FOR

- Elle permet principalement de parcourir des structures de données :
- **Syntaxe :**

for ⇔ Pour et in ⇔ dans , on peut lire : **Pour** chaque *élément* **dans** iterable

```
1 for element in iterable:  
2     ...  
3
```



# Structures itératives : Boucle FOR

- Exemples :
  - 1. afficher les nombre de 1 à 10.

```
1 for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:  
2     print(i)  
3
```

NB :

les crochets représentent une **liste** (ensemble) d'entiers.

# Structures itératives : Boucle FOR

- Autres Exemples :

```
1 for i in range(10):
2     print(i)
3 # Affiche : 0 à 9
4
5 for i in range(1, 10):
6     print(i)
7 # Affiche : 1 à 9
8
9 for i in range(1, 11):
10    print(i)
11 # Affiche : 1 à 10
12
```

# Structures itératives : Boucle FOR

- Exemples :

- 2. Afficher toutes les lettres du mot "Python" les une après les autres en majuscule.

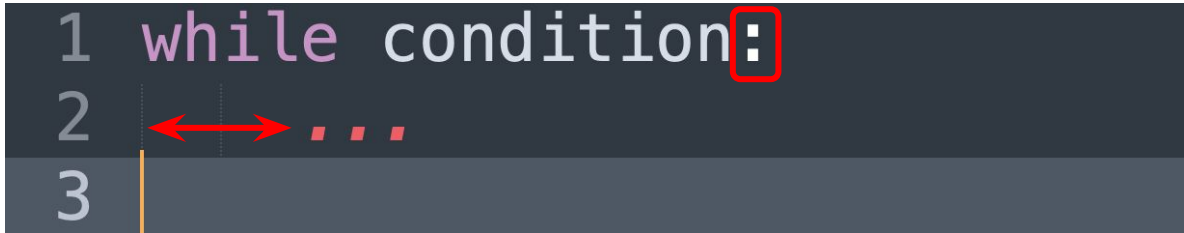
```
1 word = "Python"
2
3 for letter in word:
4     print(letter.upper())
5
6 # meme chose que :
7 for letter in "Python":
8     print(letter.upper())
9
```

# Structures itératives : Boucle WHILE

- Elle permet de faire les même tâches que la boucle **for**
- Elle permet aussi de contrôler des saisies
- **Syntaxe :**

while ⇔ Tant que, on peut lire : **Tant que** la **condition** **est** vérifiée

```
1 while condition:  
2     ...  
3
```

A code editor snippet showing the syntax of a while loop. Line 1: 'while condition:' with a red box around the colon. Line 2: '...' with a red double-headed arrow pointing from the start of the line to the ellipsis. Line 3: An empty line with a vertical cursor at the start.



# Structures itératives : Boucle WHILE

*Ne pas oublier  
d'incrémenter l'indice*

```
1 for i in range(10):  
2     print(i)  
3
```

```
5 i = 0  
6 while i < 10:  
7     print(i)  
8     i += 1
```

**Exercice** : Écrire un programme qui permet de saisir des noms puis affiche les noms saisi en majuscule. A chaque affichage on demande à l'utilisateur s'il veut continuer ou pas.

```
1 rep = "o"
2
3 while rep == "o":
4     name = input("Entrer un nom :\n")
5     print(name.upper())
6     rep = input("Continuer ? O/N\n")
7
8 print("Au revoir!!!")
9
```

# Structure itératives : ANY & ALL

Opérer sur une liste pour vérifier si **une valeur dans la liste est Vraie** ou **toutes les valeurs sont Vraies**

## ANY

`any([False, False, True, False, False, False])`



**True**

## ALL

`all([False, False, True, False, False, False])`



**False**

# Structure itératives : continue, break et pass

- **continue** : permet de sauter l'itération en cours.
- **break** : permet de sortir de la boucle.
- **pass** : cette instruction est utilisée si on veut respecter une syntaxe de code alors que rien ne sera exécuté. Elle ne fait rien en quelque sortes.

# Structure itératives : ELSE

- A la boucle **FOR**, on peut ajouter l'instruction **ELSE** qui sera exécutée si la boucle FOR a terminé toutes ces itérations.
- Ajoutée à la boucle **WHILE**, l'instruction **BREAK** est exécutée si la condition de la boucle est fausse.
- L'instruction **BREAK** fait sortir de la boucle sans exécuter l'instruction ELSE.

# Les listes

# Les listes

- Définition :

Une **liste** est une **structure de données** qui contient une série de valeurs (Ex: liste de notes, liste de courses,...). Elles sont présentes dans tous les langages de programmation quelquefois sous un autre nom.

Python autorise la construction de liste contenant des valeurs de types différents (par exemple entier et chaîne de caractères). Une liste est déclarée par une série de valeurs séparées par des **virgules**, et le tout encadré par des **crochets**.

# Les listes

- Exemples :

- liste d'entiers:

```
my-list = [1, 2, 3, 4, 5]
```

- liste de chaînes:

```
names = ["toto", "titi", "tata"]
```



# Les listes

- Remarque :

Une liste peut être vide ou contenir autant d'éléments que l'on souhaite. Dans une liste on peut avoir des éléments de type différents.

```
2
3 empty_list = []
4
5
6 items = ["Banane", False, 350, "Toto", 25.5]
7
8
9
```

# Les listes

- Ajout dans une liste :

Pour ajouter un élément dans une liste on peut utiliser :

- Le méthode `append()` :

```
2
3 my_list = []
4
5 my_list.append(10)
6
7
```

*Élément à ajouter*

**Remarque :** Cette méthode ne prend qu'un seul élément à la fois.

# Les listes

- Le méthode `extend()` :

```
3 my_list = []
4
5 my_list.extend([1,0,19,'Test'])
6
7 my_list.extend(1,0,19,'Test') Erreur!!!
8
```

*un iterable*

**NB** : On peut aussi faire des opérations sur des listes (addition, multiplication).

+ pour **fusionner** des listes et \* pour **dupliquer** les valeurs d'une liste.

# Les listes

- Exemples d'opérations :

```
1 l1 = [1,2,3]
2 l2 = [4,5,6]
3 l = l1 + l2
4 print(l) # affiche: [1,2,3,4,5,6]
```

*addition*

```
3 my_list1 = []
4 my_list2 = ["toto",18]
5 my_list1 = my_list2 * 2
6
7 print(my_list1) => ["toto",18,"toto",18]
8
```

*Multiplication*

# Les listes

- Suppression dans une liste :

Pour enlever un élément dans une liste on peut utiliser `remove()` qui enlève la première occurrence de l'élément à supprimer :

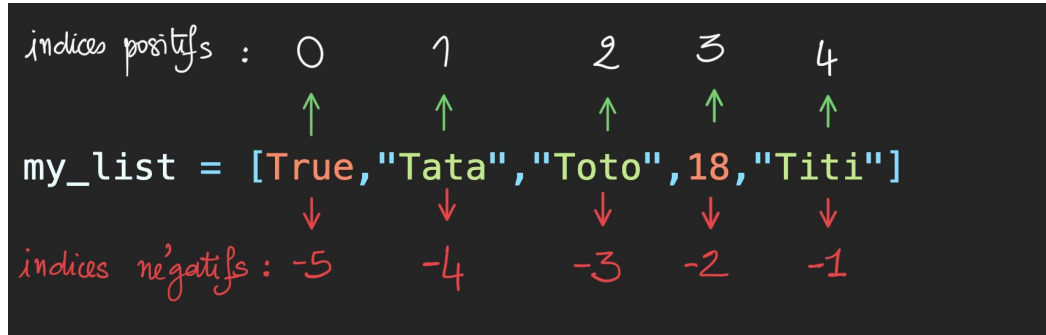
```
3 my_list = [True,"Tata","Toto",18,"Toto"]
4
5 my_list.remove("Toto")
6
7 print(my_list) ⇒ [True, "Tata", 18, "Toto"]
```

**Remarque :** Si l'élément à supprimer n'est pas dans la liste une erreur est levée.

# Les listes

- Accéder à un élément d'une liste :

Pour accéder à un élément d'une liste, on utilise son **indice**. Il représente la position de l'élément dans une liste. En python, la position du premier élément dans une liste est zéro (0). On peut aussi avoir des indices positifs ou négatifs.

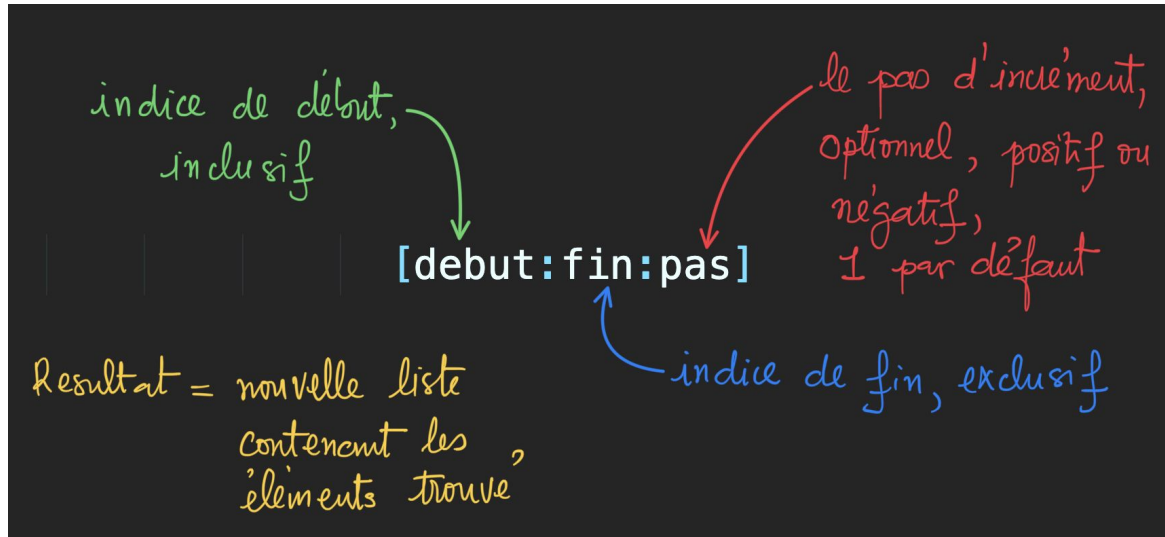


**exemples :** `my_list[0]`, `my_list[-1]`, `my_list[3]`.

# Les listes

- Récupérer une partie d'une liste :

Pour récupérer une partie d'une liste, on utilise les **SLICES (tranches)**. Ils permettent de récupérer un intervalle d'éléments d'une liste.



# Les listes

- Récupérer une partie d'une liste : EXEMPLES

```
1 cours = ["presentation","cours_1","cours_2","cours_3","cours_4"]
2
3 print(cours[:])
4 # ['presentation', 'cours_1', 'cours_2', 'cours_3', 'cours_4']
5 print(cours[:5])
6 # ['presentation', 'cours_1', 'cours_2', 'cours_3', 'cours_4']
7 print(cours[0:5])
8 # ['presentation', 'cours_1', 'cours_2', 'cours_3', 'cours_4']
9 print(cours[0:])
10 # ['presentation', 'cours_1', 'cours_2', 'cours_3', 'cours_4']
11 print(cours[0:-3])
12 # ['presentation', 'cours_1']
13 print(cours[::-1])
14 #['cours_4', 'cours_3', 'cours_2', 'cours_1', 'presentation']
15 print(cours[::-2])
16 #['cours_4', 'cours_2', 'presentation']
```



# Les listes

- Autres méthodes sur les listes (1):

- `list.index(valeur)` :

Renvoie l'index de la première occurrence de la valeur.

- `list.count(valeur)`:

Renvoie l'index de la première occurrence de la valeur.

- `list.sort() != sorted(list)` :

`sort()` permet de trier la **même liste**.

`sorted()` permet de **retourner** une nouvelle liste triée

- `list.reverse()` :

permet d'inverser les valeurs de la liste. On peut combiner `sort()` et `reverse()` pour trier dans l'ordre décroissant.

# Les listes

- Autres méthodes sur les listes (1) : EXEMPLES

```
1 names = ["toto","titi","tutu","toto","tata"]
2
3 print(names.index("titi"))
4 # 1
5 print(names.count("toto"))
6 # 2
7 names.sort()
8 print(names)
9 # ['tata', 'titi', 'toto', 'toto', 'tutu']
10 names_sorted = sorted(names)
11 print(names_sorted)
12 # ['tata', 'titi', 'toto', 'toto', 'tutu']
13 names.reverse()
14 print(names)
15 # ['tutu', 'toto', 'toto', 'titi', 'tata']
```

# Les listes

- Autres méthodes sur les listes (2):

- list.pop(index) :

Enlève la valeur se trouvant à cet index et renvoie la valeur supprimée.

- list.clear() :

Enlève tous les éléments de la liste.

- chaine.join(list) :

Joindre les éléments d'une liste sous forme de chaîne.

- list.split() ou list.split(separateur):

Créer une liste à partir d'une chaîne. Par défaut, le séparateur est l'espace.

# Les listes

- Autres méthodes sur les listes (2) : EXEMPLES

```
1 names = ["toto","titi","tutu","toto","tata"]
2
3
4
5 element_sup = names.pop(1)
6 print(element_sup)
7 # tata
8 print(names)
9 # ['toto', 'titi', 'tutu', 'toto']
10 names.clear()
11 print(names)
12 # []
```

# Les listes

- Autres méthodes sur les listes (2) : EXEMPLES

```
1 names = ["toto","titi","tutu"]
2
3 r = " ".join(names)
4 print(r)
5 # toto titi tutu
6
7 chaine = "Ceci est une chaine"
8 liste = chaine.split()
9 print(liste)
10 # ['Ceci', 'est', 'une', 'chaine']
11 liste = chaine.split("y")
12 # ['Ceci est une chaine']
```

# Les listes

- Opérateur d'appartenance (IN) :

Renvoie TRUE si une valeur donnée est présente dans une liste. Marche aussi pour les chaînes.

**Exemple :**

Supprimer une valeur dans une liste.

```
1 names = ["toto","titi","tutu","tete","tata"]
2
3 print("Affichage de liste : ")
4 print("\n".join(names))
5
6 name_remove = input("Entrer le nom à supprimer :\n")
7
8 if name_remove in names:
9     names.remove(name_remove)
10    print("Suppression reussie")
11 else:
12    print("Le nom saisi n'est pas dans la liste!")
13
```

# Les listes

- listes imbriquées:

Une liste de chaînes de caractères (les chaînes sont considérées comme une liste).

Une liste de liste

Exemple :

[{"m.aly"}]

# Les listes : compréhension de liste

Les **compréhensions de liste** ou **listes en compréhension** nous permettent **d'itérer** sur des listes et d'y filtrer des éléments grâce à l'écriture d'une seule ligne de code.

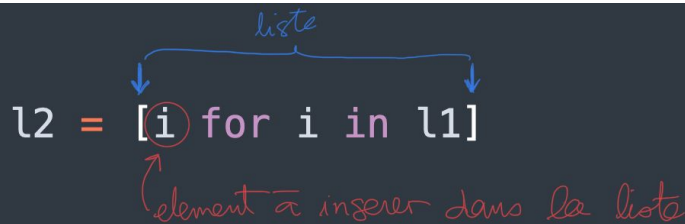
**Exemple** : Écrire un programme qui permet de copier tous les éléments d'une liste l1 vers une nouvelle liste l2.

## BOUCLE

```
l1 = [1, -2, 34, 0, 4, 69, 7, -21]
l2 = []

for i in l1:
    l2.append(i)
```

## COMPRÉHENSION DE LISTE



l2 = [i for i in l1]

The diagram shows the list comprehension syntax with handwritten annotations. A blue bracket above the 'for i in l1' part is labeled 'liste'. A red circle around the 'i' in '[i]' has a red arrow pointing to it from the text 'element à insérer dans la liste' written in red cursive below the code.



# Les listes : compréhension de liste

**Exemple 2** : Écrire un programme qui permet de copier le double de tous les nombres pairs d'une liste l1 vers une nouvelle liste l2.

**BOUCLE**

**COMPRÉHENSION DE LISTE**