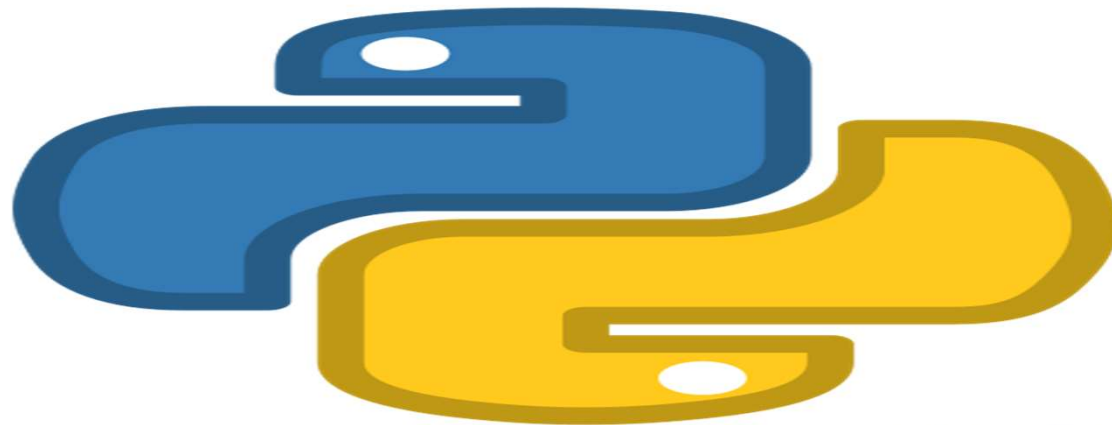


# Python : fondamentaux

**Mohammed OUANAN**

[m.ouanan@umi.ac.ma](mailto:m.ouanan@umi.ac.ma)



# Plan

- 1 Variables
  - Déclaration
  - Type
  - Alias
  - Identité
  - Opérations sur les variables
  - Suppression

- 2 Lecture d'une saisie

- 3 Fichiers de code Python

# Plan

## 4 Formatage de chaîne de caractère

## 5 Commentaires

## 6 Structures conditionnelles

- if
- if ... else
- if ... elif ... else
- pass
- match (switch)

# Plan

## 7 Structures itératives

- while
- while ... break
- while ... continue
- while ... else
- for
- for-one-line
- for ... break ... else
- Utilisation d'underscore

## 8 Portée d'une variable

## 9 Constantes

# Python

## Une variable ?

- Un pointeur vers une zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

# Python

## Une variable ?

- Un pointeur vers une zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

## Python est un langage de programmation fortement typé, mais

- Pas de type à préciser pour les variables
- Une variable peut changer de type
- Impossible de concaténer une chaîne et un nombre sans conversion

# Python

## Caractéristiques d'une variable

- Son nom dans le programme
- Son type (déterminant souvent l'espace mémoire réservé pour la variable)
- Son identité

# Python

## Conventions de nommage des variables en **Python**

- Choisir des noms courts et significatifs
- Utiliser **snake\_case**
- Éviter les conflits avec les mots-clés de **Python** tels que `if`, `for`, `class`, `str`...
- Utiliser des noms explicites pour les booléens comme `est_majeur`...
- Éviter les caractères spéciaux (Bien que **Python** 3 permette l'utilisation de caractères non ASCII, il est généralement préférable d'éviter les lettres accentuées et autres caractères spéciaux)



# Python

## Pour déclarer une variable

```
>>> x = 2
```

# Python

## Pour déclarer une variable

```
>>> x = 2
```

## Pour afficher une variable

```
>>> x  
2
```

# Python

## Pour déclarer une variable

```
>>> x = 2
```

## Pour afficher une variable

```
>>> x  
2
```

## Pour utiliser une variable

```
>>> x + 3  
5
```

# Python

**Pour déclarer plusieurs variables et leur affecter une même valeur**

```
>>> a = b = c = 1
```

# Python

**Pour déclarer plusieurs variables et leur affecter une même valeur**

```
>>> a = b = c = 1
```

**Pour afficher les valeurs respectives des variables précédentes**

```
>>> a, b, c  
(1, 1, 1)
```

# Python

**Pour déclarer plusieurs variables et leur affecter une même valeur**

```
>>> a = b = c = 1
```

**Pour afficher les valeurs respectives des variables précédentes**

```
>>> a, b, c  
(1, 1, 1)
```

**Pour déclarer plusieurs variables et leur affecter des valeurs différentes**

```
>>> d, e, f = 1, 2, "bonjour"
```

# Python

**PEP 15 recommande de séparer les groupes de 3 chiffres par \_**

```
x = 1_000_000
```

# Python

**PEP 15 recommande de séparer les groupes de 3 chiffres par \_**

```
x = 1_000_000
```

## Résultat

```
print(x)  
#affiche  
1000000
```



# Python

## Remarques

- Les variables non-initialisées n'ont pas une valeur par défaut.
- L'accès à une variable non-initialisée déclenche une erreur de `type NameError`.

# Python

**Depuis Python 3.5, une variable peut être typée**

```
>>> x: int = 2
```

# Python

**Depuis Python 3.5, une variable peut être typée**

```
>>> x: int = 2
```

**Aucune vérification de type par Python, donc  $x$  peut être réaffectée sans respecter le type initial**

```
>>> x = 'bonjour'
```

# Python

Depuis Python 3.5, une variable peut être typée

```
>>> x: int = 2
```

Aucune vérification de type par Python, donc `x` peut être réaffectée sans respecter le type initial

```
>>> x = 'bonjour'
```

Pour déterminer le type d'une variable

```
>>> type(x)  
<class 'str'>
```

# Python

## Quelques types de variable selon la valeur affectée

- Booléen : `bool`
- Numérique : `int`, `float`, `complex`
- Texte : `str`
- Binaires : `bytes`, `bytearray`, `memoryview`
- Multi-valeurs : `array`, `list`, `tuple`
- Ensemble : `set`
- Ensemble clé-valeur : `dict`

# Python

## Exemple avec les booléens

```
>>> type(True)
<class 'bool'>
```

# Python

## Exemple avec les booléens

```
>>> type(True)
<class 'bool'>
```

## Exemple avec les nombres réels

```
>>> type(2.8)
<class 'float'>
```

# Python

## Exemple avec les booléens

```
>>> type(True)
<class 'bool'>
```

## Exemple avec les nombres réels

```
>>> type(2.8)
<class 'float'>
```

## Exemple avec les nombres complexes

```
>>> type(2j)
<class 'complex'>
```



# Python

**Pour récupérer le nom du type sous forme d'une chaîne de caractères**

```
>>> val = 2
>>> val.__class__.__name__
'int'
>>> type(val).__name__
'int'
```

# Python

**Pour tester si une valeur est de certain type**

```
>>> isinstance(True, bool)
True
```

# Python

**Pour tester si une valeur est de certain type**

```
>>> isinstance(True, bool)
True
```

## Exemple 2

```
>>> isinstance(2.5, int)
False
```

# Python

## Pour tester si une valeur est de certain type

```
>>> isinstance(True, bool)
True
```

### Exemple 2

```
>>> isinstance(2.5, int)
False
```

### Exemple 3

```
>>> isinstance(2, float)
False
```

# Python

**Le deuxième paramètre peut être un tuple**

```
>>> isinstance(2.5, (int, float))  
True
```

# Python

Le deuxième paramètre peut être un tuple

```
>>> isinstance(2.5, (int, float))  
True
```

Exemple 2

```
>>> isinstance("bonjour", (int, float))  
False
```

# Python

**Il est possible de définir un alias de type**

```
>>> type number = int | float
```

# Python

**Il est possible de définir un alias de type**

```
>>> type number = int | float
```

**Aucune vérification de type par Python, donc  $x$  peut être réaffectée sans respecter le type initial**

```
>>> x: number = 0
```



# Python

Il est possible de définir un alias de type

```
>>> type number = int | float
```

Aucune vérification de type par Python, donc `x` peut être réaffectée sans respecter le type initial

```
>>> x: number = 0
```

Pour déterminer le type d'une variable

```
>>> type(x).__name__  
int
```

# Python

## Identité d'une variable

- Constante (de type entier) unique pour chaque variable (objet)
- Deux variables ayant la même identité = deux variables partageant la même valeur
- Certaines variables changent d'identité en changeant de valeur (immuable ou immutable) et certaines autres gardent leur identité (muable ou mutable)

# Python

## Exemple

```
>>> x=2
>>> y=2
>>> x,y
(2, 2)
>>> id(x) , id(y)
(1571223488, 1571223488)
```

# Python

## Exemple

```
>>> x=2
>>> y=2
>>> x,y
(2, 2)
>>> id(x) , id(y)
(1571223488, 1571223488)
```

Pour récupérer la représentation hexadécimale de cet identifiant entier

```
>>> hex(id(x))
```

# Python

## Opérations sur les nombres

- `=` : affectation
- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `**` : puissance
- `/` : division
- `//` : division entière
- `%` : reste de la division

# Python

## Exemples

```
>>> 5+2
```

```
7
```

```
>>> 5*2
```

```
10
```

```
>>> 5/2
```

```
2.5
```

```
>>> 5//2
```

```
2
```

```
>>> 5**2
```

```
25
```

```
>>> 5%2
```

```
1
```

```
>>>
```

# Python

## Quelques opérations sur les variables numériques

•  $i += 2 \Rightarrow i = i + 2$

•  $i -= 2 \Rightarrow i = i - 2$

•  $i *= 2 \Rightarrow i = i * 2$

•  $i /= 2 \Rightarrow i = i / 2$

•  $i \% = 2 \Rightarrow i = i \% 2$

•  $i //= 2 \Rightarrow i = i // 2$

•  $i ** = 2 \Rightarrow i = i ** 2$

• ...

# Python

## Exercice

Écrire un script **Python** qui permet de permuter le contenu de deux variables sans utiliser de variable temporaire.



# Python

## Première solution

```
>>> a=2
>>> b=3
>>> a+=b
>>> b=a-b
>>> a-=b
>>> a,b
(3, 2)
```

# Python

## Première solution

```
>>> a=2
>>> b=3
>>> a+=b
>>> b=a-b
>>> a-=b
>>> a,b
(3, 2)
```

## Deuxième solution (on peut également entourer $a, b$ par des `()` ou `[]`)

```
>>> a=2
>>> b=3
>>> a,b = b,a
>>> a,b
(3, 2)
```

**Considérons les deux chaînes de caractères suivantes**

```
str1, str2 = 'bon', "jour"
```

**Considérons les deux chaînes de caractères suivantes**

```
str1, str2 = 'bon', "jour"
```

**Utilisons l'opérateur + pour concaténer deux chaînes de caractère**

```
>>> str3 = str1 + str2  
>>> str3  
'bonjour'
```

**Considérons les deux chaînes de caractères suivantes**

```
str1, str2 = 'bon', "jour"
```

**Utilisons l'opérateur + pour concaténer deux chaînes de caractère**

```
>>> str3 = str1 + str2  
>>> str3  
'bonjour'
```

**Utilisons l'opérateur \* pour répéter un motif plusieurs fois**

```
>>> str3 * 4  
'bonjourbonjourbonjourbonjour'
```

Considérons les deux chaînes de caractères suivantes

```
str1, str2 = 'bon', "jour"
```

Utilisons l'opérateur + pour concaténer deux chaînes de caractère

```
>>> str3 = str1 + str2  
>>> str3  
'bonjour'
```

Utilisons l'opérateur \* pour répéter un motif plusieurs fois

```
>>> str3 * 4  
'bonjourbonjourbonjourbonjour'
```

**Multiplier une chaîne de caractères par une deuxième chaîne déclenche une erreur**

```
>>> str3 * "4"  
TypeError: can't multiply sequence by non-int of type 'str'
```

# Python

**Pour créer une chaîne de caractères dont la valeur est définie sur plusieurs lignes**

```
>>> msg='hello\  
... world'  
>>> msg  
'helloworld'
```

# Python

Pour créer une chaîne de caractères dont la valeur est définie sur plusieurs lignes

```
>>> msg='hello\  
... world'  
>>> msg  
'helloworld'
```

On peut également utiliser les `"""`

```
>>> msg2="""bonjour  
... tout  
... le  
... monde"""  
>>> msg2  
'bonjour\ntout\nle\nmonde'
```



# Python

**Pour accéder à un caractère à la position 2 (premier caractère d'indice 0)**

```
>>> str3[2]  
'n'
```

# Python

**Pour accéder à un caractère à la position 2 (premier caractère d'indice 0)**

```
>>> str3[2]  
'n'
```

**Pour extraire les caractères entre la position 2 et 4**

```
>>> str3[2:4]  
'nj'
```

# Python

**Pour extraire les caractères de la position 2 jusqu'à la fin**

```
>>> str3[2:]  
'njour'
```

# Python

**Pour extraire les caractères de la position 2 jusqu'à la fin**

```
>>> str3[2:]  
'njour'
```

**Pour extraire les caractères du début jusqu'à la position 4**

```
>>> str3[:4]  
'bonj'
```

# Python

**Pour extraire les caractères de la position 2 jusqu'à la fin**

```
>>> str3[2:]  
'njour'
```

**Pour extraire les caractères du début jusqu'à la position 4**

```
>>> str3[:4]  
'bonj'
```

**Pour extraire les caractères entre la position 2 et 4 en commençant de la fin**

```
>>> str3[-4:-2]  
'jo'
```

# Python

## Pour récupérer la taille d'une chaîne de caractères

```
>>> len(str3)  
7
```

# Python

## Pour récupérer la taille d'une chaîne de caractères

```
>>> len(str3)  
7
```

## Pour remplacer une partie de la chaîne par une autre

```
>>> str3.replace('jour', 'soir')  
'bonsoir'
```

# Python

## Pour récupérer la taille d'une chaîne de caractères

```
>>> len(str3)
7
```

## Pour remplacer une partie de la chaîne par une autre

```
>>> str3.replace('jour', 'soir')
'bonsoir'
```

## Pour découper une chaîne de caractère selon un séparateur précisé

```
>>> str3.split('o')
['b', 'nj', 'ur']
```



# Python

## Quelques méthodes sur les chaînes de caractères

- `capitalize()` : convertit la première lettre du premier mot en majuscule
- `title()` : convertit la première lettre de chaque mot en majuscule
- `index(x)` : retourne l'indice de la première occurrence de `x` dans la chaîne, erreur sinon. (`rindex` pour la dernière occurrence)
- `find(x)` : retourne l'indice de la première occurrence de `x` dans la chaîne, -1 sinon. (`rfind` pour la dernière occurrence)
- `upper()` : convertit une chaîne de caractères en majuscule
- `lower()` : convertit une chaîne de caractères en minuscule
- `strip()` : supprime les espaces au début et à la fin (autres variations `rstrip` et `lstrip`)
- `startswith(x)` : retourne `true` si la chaîne commence par `x`, `false` sinon.
- `endswith(x)` : retourne `true` si la chaîne se termine par `x`, `false` sinon.
- `count(x)` : retourne le nombre d'occurrence de `x` dans la chaîne
- ...

# Python

## Les méthodes `isX()` : retourne un booléen

- `islower()` : vérifie si tous les caractères d'une chaîne sont en minuscule
- `isupper()` : vérifie si tous les caractères d'une chaîne sont en majuscule
- `isalnum()` : vérifie si tous les caractères d'une chaîne sont alphanumériques
- `isalpha()` : vérifie si tous les caractères d'une chaîne sont alphabétiques
- `isnumeric()` : vérifie si tous les caractères d'une chaîne sont numériques
- `isspace()` : vérifie si tous les caractères d'une chaîne sont des espaces
- ...

# Python

**Pour certains opérateurs, ceci génère une erreur**

```
>>> 2 + '3'  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

# Python

**Pour certains opérateurs, ceci génère une erreur**

```
>>> 2 + '3'  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

**Pour additionner un nombre et une chaîne de caractère, on peut convertir la chaîne en entier**

```
>>> 2 + int('3')  
5
```

# Python

**Pour certains opérateurs, ceci génère une erreur**

```
>>> 2 + '3'  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

**Pour additionner un nombre et une chaîne de caractère, on peut convertir la chaîne en entier**

```
>>> 2 + int('3')  
5
```

**Pour faire la concaténation**

```
>>> str(2) + '3'  
'23'
```

# Python

Dans un contexte arithmétique, `True` vaut 1

```
>>> int(True)  
1
```

# Python

Dans un contexte arithmétique, `True` vaut 1

```
>>> int(True)  
1
```

Et

```
>>> 1 + True  
2
```

# Python

Dans un contexte arithmétique, `True` vaut 1

```
>>> int(True)  
1
```

Et

```
>>> 1 + True  
2
```

`False` vaut 0

```
>>> int(False)  
0
```



# Python

## Quelques fonctions de conversion en Python

- `float()` : convertit en réel la valeur passée en paramètre si elle est numérique, une erreur sera générée sinon.
- `int()` : convertit en entier la valeur passée en paramètre si elle est numérique, une erreur sera générée sinon.
- `str()` : convertir en chaîne de caractère la valeur passée en paramètre
- `bool(x)` : retourne `False` si `x` contient le nombre 0, [], (), {} ou `False`, `True` sinon.
- ...

# Python

## Pour supprimer une variable

```
>>> del x
```

# Python

## Pour supprimer une variable

```
>>> del x
```

## Demander la variable d'une variable supprimée génère une erreur

```
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

# Python

**Pour lire une saisie de console (une chaîne de caractère)**

```
>>> x = input()
```

# Python

**Pour lire une saisie de console (une chaîne de caractère)**

```
>>> x = input()
```

**On peut ajouter un message indicatif**

```
>>> x = input('Quel est votre nom ?\n')
```

# Python

**Pour lire une saisie de console (une chaîne de caractère)**

```
>>> x = input()
```

**On peut ajouter un message indicatif**

```
>>> x = input('Quel est votre nom ?\n')
```

**Pour lire une saisie numérique (entier par exemple)**

```
>>> num1 = int(input('saisir un nombre'))
```

# Python

## Fichiers de code **Python** : idée

- Créer un projet **Python** (dossier contenant des fichiers avec l'extension `.py`)
- Utiliser un IDE pour écrire le code **Python**
- Utiliser la console **Python** pour exécuter le contenu du projet

# Python

## Quel IDE (Environnement de développement intégré) ?

- **Visual Studio Code** (À ne pas confondre avec Visual Studio)
- PyCharm
- Eclipse
- ...



# Python

## Visual Studio Code : téléchargement

[code.visualstudio.com/download](https://code.visualstudio.com/download)

# Python

## Visual Studio Code : téléchargement

[code.visualstudio.com/download](https://code.visualstudio.com/download)

## Visual Studio Code (ou VSC) , pourquoi ?

- Gratuit
- Multi-langage
- Multi-système d'exploitation
- Extensible via l'installation de quelques extensions

# Python

## Quelques raccourcis VSC

- Pour activer la sauvegarde automatique : aller dans `File > AutoSave`
- Pour indenter son code : `Alt` `Shift` `f`
- Pour commenter/décommenter : `Ctrl` `:`
- Pour sélectionner toutes les occurrences : `Ctrl` `f2`
- Pour sélectionner l'occurrence suivante : `Ctrl` `d`
- Pour placer le curseur dans plusieurs endroits différents : `Alt`

# Python

## Quelques extensions VSC pour Python

- **Python** : pour l'intelliSense(pour l'autocomplétion et la suggestion de code)
- **autopep8** : pour l'indentation et le formatage
- **autoDocstring - Python Docstring Generator** : pour la génération de la documentation (DocString)
- **Code Runner** : pour l'ajout d'un bouton d'exécution/interprétation

# Python

## Pour créer un projet sous VSC

- Allez dans `File > Open Folder...`
- Cliquez sur `Nouveau dossier` **et saisissez** `cours_python`
- Cliquez sur le dossier `cours_python` puis sur le bouton `Sélectionner un dossier`
- Créez un fichier `main.py` dans `cours_python`

# Python

## Contenu de `main.py`

```
msg = "Hello World"  
print(msg)
```

# Python

## Contenu de `main.py`

```
msg = "Hello World"  
print(msg)
```

## Pour exécuter

cliquez sur le triangle ▷ (en haut à droite)

# Python

## Contenu de `main.py`

```
msg = "Hello World"  
print(msg)
```

## Pour exécuter

cliquez sur le triangle ▷ (en haut à droite)

**Ou exécutez la commande suivante depuis un terminal**

```
python main.py
```



# Python

## Exercice

Écrire un script **Python** qui

- demande à l'utilisateur de saisir deux nombres
- affiche le résultat de la somme comme suit :  
la somme de 2 et 3 est : 5
- 2 et 3 étant les valeurs saisies par l'utilisateur

# Python

**Une première solution consiste à faire**

```
x = int(input('Première valeur '))  
y = int(input('Deuxième valeur '))  
resultat = x + y  
print('la somme de', x, 'et', y, 'est :', resultat)
```

# Python

Une première solution consiste à faire

```
x = int(input('Première valeur '))  
y = int(input('Deuxième valeur '))  
resultat = x + y  
print('la somme de', x, 'et', y, 'est :', resultat)
```

## Remarque

**Python** ajoute un espace après chaque paramètre de la fonction `print()`.

# Python

Pour modifier le séparateur, on utilise l'option `sep` (ici c'est `-`)

```
x = int(input('Première valeur '))
y = int(input('Deuxième valeur '))
resultat = x + y
print('la somme de', x, 'et', y, 'est :', resultat,
      sep="-")
```

# Python

**Solution avec le formatage de la chaîne de sortie en utilisant % comme en langage C**

```
x = int(input('Première valeur '))  
y = int(input('Deuxième valeur '))  
resultat = 'la somme de %d et %d est : %d'  
print(resultat % (x, y, x + y))
```

# Python

## Solution avec le formatage de la chaîne de sortie en utilisant % comme en langage C

```
x = int(input('Première valeur '))
y = int(input('Deuxième valeur '))
resultat = 'la somme de %d et %d est : %d'
print(resultat % (x, y, x + y))
```

### Autres options

- %s pour str
- %f pour float
- ...

# Python

## Solution avec le formatage de la chaîne de sortie (disponible depuis Python 2.6)

```
x = int(input('Première valeur '))  
y = int(input('Deuxième valeur '))  
resultat = 'la somme de {} et {} est : {}'  
print(resultat.format(x, y, x + y))
```

# Python

## Solution avec le formatage de la chaîne de sortie (disponible depuis Python 2.6)

```
x = int(input('Première valeur '))
y = int(input('Deuxième valeur '))
resultat = 'la somme de {} et {} est : {}'.format(x, y, x + y)
print(resultat)
```

## Un raccourci de l'écriture précédente (disponible depuis Python 3.6 : f-string)

```
x = int(input('Première valeur '))
y = int(input('Deuxième valeur '))
print(f'la somme de {x} et {y} est : {x + y}')
```



# Python

## Pour formater l'affichage d'un nombre réel

```
float = 12345.12345  
print(f'Le nombre après formatage : {float:.2f}' )  
# affiche 12345.12
```

# Python

## Pour formater l'affichage d'un nombre réel

```
float = 12345.12345  
print(f'Le nombre après formatage : {float:.2f}' )  
# affiche 12345.12
```

## Ce qui est interdit dans les `{ }` d'une f-string

- le # des commentaires
- le antislash
- ...

# Python

## Commentaire

texte ignoré par le compilateur

# Python

## Commentaire

texte ignoré par le compilateur

## Syntaxe

```
# le texte suivant sera ignoré
```

# Python

## Commentaire

texte ignoré par le compilateur

## Syntaxe

```
# le texte suivant sera ignoré
```

## Remarque

**Python** n'a pas de symbole particulier pour les commentaires multi-lignes

# Python

## Commentaire de documentation (**DocString**)

- pouvant être récupéré
- pouvant être multi-lignes
- généralement attaché à un élément **Python**

# Python

## Commentaire de documentation (**DocString**)

- pouvant être récupéré
- pouvant être multi-lignes
- généralement attaché à un élément **Python**

## Syntaxe

```
"""message sur plusieurs lignes  
attaché à un élément  
décrivant un élément Python"""
```

# Python

## Structures conditionnelles

- `if ... [[elif ...] else ...]`
- `match ... case` depuis **Python 3.10**



# Python

## Exécuter si une condition est vraie

```
if (condition[s]):  
    # instruction[s]
```

# Python

## Exécuter si une condition est vraie

```
if (condition[s]) :  
    # instruction[s]
```

## Remarques

- Les parenthèses ne sont pas obligatoires
- L'indentation est obligatoire pour les instructions de `if`
- Pour les conditions, on utilise les opérateurs de comparaison

# Python

## Exemple

```
x = 10
if (x > 0):
    print(str(x) + " est positif")
```

# Python

## Exemple

```
x = 10
if (x > 0):
    print(str(x) + " est positif")
```

## One-line if

```
x = 10
if (x > 0): print(str(x) + " est positif")
```

# Python

## Opérateurs de comparaison

- `==` : pour tester l'égalité
- `!=` : pour tester l'inégalité
- `>` : supérieur à
- `<` : inférieur à
- `>=` : supérieur ou égal à
- `<=` : inférieur ou égal à

# Python

## Opérateurs de comparaison

- == : pour tester l'égalité
- != : pour tester l'inégalité
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

**Python** n'autorise pas de comparer deux valeurs de type incompatible.

# Python

## Remarques

- En **Python**, les valeurs suivantes sont considérées comme `falsey` lorsqu'elles sont évaluées dans un contexte booléen :
  - `False` : la valeur booléenne.
  - `None` : la valeur nulle.
  - `0` : la valeur entière zéro.
  - `0.0` : la valeur flottante zéro.
  - `''` : la chaîne de caractères vide.
  - `[]` : la liste vide.
  - `{}` : le dictionnaire vide.
  - `()` : le tuple vide.
- Toutes les autres valeurs sont considérées comme `truthy` lorsqu'elles sont évaluées dans un contexte booléen, ce qui signifie qu'elles sont considérées comme vraies.

# Python

**Voici un exemple de code illustrant cela**

```
falsy_values = [False, None, 0, 0.0, '', [], {}]

for value in falsy_values:
    if value:
        print(f"{value} est truthy")
    else:
        print(f"{value} est falsy")
```



# Python

**Voici un exemple de code illustrant cela**

```
falsy_values = [False, None, 0, 0.0, '', [], {}]

for value in falsy_values:
    if value:
        print(f"{value} est truthy")
    else:
        print(f"{value} est falsy")
```

**Voici un exemple de code illustrant cela**

```
False est falsy
None est falsy
0 est falsy
0.0 est falsy
'' est falsy
[] est falsy
{} est falsy
```

# Python

Cependant, une valeur `falsy` n'est pas forcément égale à `False`

```
print("" == False)  
# affiche False
```

# Python

## Exercice

Écrire un code **Python** qui demande à l'utilisateur de saisir un entier positif et qui affiche ensuite sa parité (sans `else`).

# Python

## Opérateurs logiques

- `and` : **et**
- `or` : **ou**
- `not` : **non**
- `^` : **ou exclusif**

# Python

## Opérateurs logiques

- `and` : et
- `or` : ou
- `not` : non
- `^` : ou exclusif

**Tester plusieurs conditions (en utilisant des opérateurs logiques)**

```
if (condition1 and condition2 or condition3):  
    # instructions
```

# Python

## Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if 2 > 1 and 3 > 2:  
    print(True)
```

```
# affiche True
```

# Python

## Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if 2 > 1 and 3 > 2:  
    print(True)
```

```
# affiche True
```

## Ou en plus simple

```
if 3 > 2 > 1:  
    print(True)
```

```
# affiche True
```

# Python

**Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`)**

```
if (condition[s]):  
    # instruction[s]  
else:  
    # instruction[s]
```



## Exemple

```
x = -10
if (x > 0):
    print(str(x) + " est positif")
else:
    print(str(x) + " est n'egatif")
```

## Exemple

```
x = -10
if (x > 0):
    print(str(x) + " est positif")
else:
    print(str(x) + " est n'egatif")
```

## One-line if else

```
print(str(x) + " est positif") if (x > 0) else print(str(x) + " est n'e  
gatif")
```

## Exemple

```
x = -10
if (x > 0):
    print(str(x) + " est positif")
else:
    print(str(x) + " est n'egatif")
```

## One-line if else

```
print(str(x) + " est positif") if (x > 0) else print(str(x) + " est n'e  
gatif")
```

## Ou en plus simple

```
print(str(x) + (" est positif" if (x > 0) else " est n'egatif"))
```

## Exemple

```
x = -10
if (x > 0):
    print(str(x) + " est positif")
else:
    print(str(x) + " est négatif")
```

## One-line if else

```
print(str(x) + " est positif") if (x > 0) else print(str(x) + " est négatif")
```

## Ou en plus simple

```
print(str(x) + (" est positif" if (x > 0) else " est négatif"))
```

## Ou en utilisant un opérateur ternaire

```
print(str(x) + (" est négatif", " est positif")[x > 0])
```

# Python

## Exercice

Écrire un programme **Python** qui permet de déterminer si une chaîne de caractères saisie par l'utilisateur contient un nombre pair ou impair de caractères.

# Python

On peut enchaîner les conditions avec `elif` (et avoir un troisième bloc voire ... un nième)

```
if (condition[s]):  
    # instruction[s]  
elif (condition[s]):  
    # instruction[s]  
...  
else:  
    # instruction[s]
```

## Exemple

```
x = 0
if (x > 0):
    print(str(x) + " est positif")
elif (x == 0):
    print(str(x) + " est nul")
else:
    print(str(x) + " est n'egatif")
```

## Exemple

```
x = 0
if (x > 0):
    print(str(x) + " est positif")
elif (x == 0):
    print(str(x) + " est nul")
else:
    print(str(x) + " est négatif")
```

## One-line if elif else

```
x = 0
print(str(x) + " est positif") if (x > 0) else print(str(x) + " est nul")
if x == 0 else print(str(x) + " est négatif")
```



## Exemple

```
x = 0
if (x > 0):
    print(str(x) + " est positif")
elif (x == 0):
    print(str(x) + " est nul")
else:
    print(str(x) + " est négatif")
```

## One-line if elif else

```
x = 0
print(str(x) + " est positif" if (x > 0) else print(str(x) + " est nul")
      if x == 0 else print(str(x) + " est négatif"))
```

## Ou en plus simple

```
x = 0
print(str(x) + (" est positif" if (x > 0) else " est nul" if x == 0
               else " est négatif"))
```

# Python

## Exercice 1

Écrire un code **Python** qui

- demande à l'utilisateur de saisir une année (un entier),
- affiche si l'année saisie est bissextile (voir [https://fr.wikipedia.org/wiki/Ann%C3%A9e\\_bissextile](https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile)).

# Python

## Exercice 2

Écrire un code **Python** qui

- demande à l'utilisateur de saisir deux entiers  $a$  et  $b$  différents de zéro,
- affiche le signe du résultat de la multiplication sans calculer le produit.

# Python

## Exercice 3

Écrire un code **Python** qui

- demande à l'utilisateur de saisir deux entiers  $a$  et  $b$ ,
- détermine et affiche si le résultat de l'addition (sans calculer la somme) est pair ou impair.

# Python

**Si, pour certains conditions, on n'a rien à faire, on peut utiliser `pass`**

```
x = 0
if (x > 0):
    print(str(x) + " est positif")
elif (x == 0):
    pass
else:
    print(str(x) + " est négatif")
```

# Python

## Structure conditionnelle `match` : syntaxe

```
match nomVariable:  
    case constante-1:  
        groupe-instructions-1  
  
    case constante-2:  
        groupe-instructions-2  
  
    ...  
  
    case constante-N:  
        groupe-instructions-N  
  
    case default:  
        groupe-instructions-par-défaut;
```

# Python

## Remarques

- Le `match` permet **seulement** de tester l'égalité et l'inégalité.
- Le `break` est implicite.
- Il est possible de regrouper plusieurs `case`.
- Le bloc `default` est facultatif, il sera exécuté si la valeur de la variable ne correspond à aucune constante de `case`.

# Python

## Structure conditionnelle avec `match`

```
x = 2
match (x):
    case 1:
        print('un')
    case 2:
        print("deux")
    case 3:
        print("trois")
    case default:
        print("autre")

# affiche 2
```



# Python

## Remarque

case default peut être remplacé par case other ou case \_.  
Aucune différence entre les 3.

# Python

match **compare le type et la valeur**

```
x = 2
match x:
    case 1:
        print('un')
    case '2':
        print("deux")
    case 3:
        print("trois")
    case default:
        print("autre")

# affiche autre
```

# Python

Il est possible de regrouper plusieurs valeurs dans un même `case`

```
x = 2
match x:
    case 1 | 2 | 3:
        print('un, deux ou trois')
    case 4 | 5:
        print("quatre ou 5")
    case 6:
        print("six")
    case default:
        print("autre")

# affiche un, deux ou trois
```

# Python

Un `match` permet de tester l'égalité et aussi l'inégalité

```
age = 20
```

```
match age:
    case a if a < 18:
        print("Personne mineure")
    case a if a >= 18 and a < 65:
        print("Personne adulte")
    case _:
        print("Personne âgée")
```

```
# affiche adulte
```

# Python

Un `match` peut accepter une ou plusieurs valeurs

```
x, y = 2, 5

match x, y:
    case (0, 0):
        print("Les deux valeurs sont nulles")
    case a, b if a > 0 and b > 0:
        print("x et y sont positifs")
    case a, b if a < 0 or b < 0:
        print("Au moins l'une des valeurs est négative")
    case _:
        print("Au moins l'une des valeurs est nulle")

# affiche x et y sont positifs
```

# Python

## Exercice

Écrire un programme qui demande à l'utilisateur de saisir l'indice d'un mois (entier compris entre 1 et 12) et qui retourne le nombre de jours de ce mois

- si l'entier est égal à 1, 3, 5, 7, 8, 10 ou 12 le programme affiche 31
- sinon si l'entier est égal à 4, 6, 9 ou 11 le programme affiche 30
- sinon si l'entier est égal à 2, le programme demande à l'utilisateur de saisir l'année et lui retourne 29 si l'année est bissextile, 28 sinon (voir [https://fr.wikipedia.org/wiki/Ann%C3%A9e\\_bissextile](https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile))
- pour toute autre valeur, le programme affiche une erreur

# Python

**Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements**

```
while condition[s]:  
    # instruction[s]
```

# Python

**Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements**

```
while condition[s]:  
    # instruction[s]
```

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.



# Python

## Structures itératives

- `while` : pour un nombre d'itérations connu ou inconnu
- `for` : pour un nombre d'itérations connu
- Pas de `do ... while` en **Python**

# Python

## Exemple

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

# Python

## Exemple

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

## Le résultat est

```
0
1
2
3
4
```

# Python

## Pour tout afficher sur une seule ligne

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

# Python

**Pour tout afficher sur une seule ligne**

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

**Le résultat est**

```
0 1 2 3 4
```

# Python

## Exercice 1

Écrire un script **Python** qui permet d'afficher les nombres pairs inférieurs à 10.

# Python

## Exercice 2

Écrire un script **Python** qui

- demande à l'utilisateur de saisir une chaîne de caractères
- compte puis affiche le nombre de voyelles définies dans la chaîne saisie

# Python

## Solution

```
texte = input('Votre texte ')
voyelles = 'aeouiy'
i = 0
nbr_voyelles = 0
while i < len(texte):
    if texte[i] in voyelles:
        nbr_voyelles += 1
    i += 1
print(f'#nombre_voyelles = {nbr_voyelles}')
```



# Python

## Exemple avec break

```
i = 0
while i < 5:
    print(i)
    if i == 2:
        break
    i += 1
```

# Python

## Exemple avec break

```
i = 0
while i < 5:
    print(i)
    if i == 2:
        break
    i += 1
```

## Le résultat est

```
0
1
2
```

# Python

## Exemple avec `continue`

```
i = 0
while i < 5:
    i += 1
    if i == 2:
        continue
    print(i)
```

# Python

## Exemple avec continue

```
i = 0
while i < 5:
    i += 1
    if i == 2:
        continue
    print(i)
```

## Le résultat est

```
1
3
4
5
```

# Python

```
while ... else
```

- La clause `else` n'est exécutée que lorsque la condition de `while` devient fausse.
- Si on sort de la boucle ou si une exception est levée, le bloc `else` ne sera pas exécuté.

# Python

## Exemple avec `while ... else`

```
i = 0
while i < 5:
    i += 1
    print(i)
else:
    print("i est égal à 5")
```

# Python

## Exemple avec `while ... else`

```
i = 0
while i < 5:
    i += 1
    print(i)
else:
    print("i est égal à 5")
```

## Le résultat est

```
1
2
3
4
5
i est égal à 5
```

# Python

## Exercice

Écrire un script **Python** qui

- demande à l'utilisateur de saisir une chaîne de caractères
- vérifie si le texte saisi contient une voyelle (ou pas)



# Python

## Solution

```
texte = input('Votre texte ')
voyelles = 'aeouiy'
i = 0
while i < len(texte):
    if texte[i].lower() in voyelles:
        print(f'{texte} contient au moins une voyelle : {texte[i]}')
        break
    i += 1
else:
    print(f'{texte} ne contient aucune voyelle')
```

# Python

## Boucle for

```
for elt in iterable:  
    # instruction[s]
```

# Python

## Boucle for

```
for elt in iterable:  
    # instruction[s]
```

Une boucle `for` permet d'itérer sur les structures suivantes

- List
- Tuple
- Set
- Dictionary
- String
- Range
- Generator expression

# Python

## Pour parcourir une chaîne de caractères

```
chaine = "bonjour"
```

```
for elt in chaine:  
    print(elt)
```

# Python

## Pour parcourir une chaîne de caractères

```
chaine = "bonjour"  
  
for elt in chaine:  
    print(elt)
```

## Le résultat est

```
b  
o  
n  
j  
o  
u  
r
```

# Python

Et si on a besoin d'afficher des valeurs (de 0 à 4 par exemple) sans avoir d'itérable, on utilise la fonction `range`

```
for i in range(5):  
    print(i)
```

# Python

Et si on a besoin d'afficher des valeurs (de 0 à 4 par exemple) sans avoir d'itérable, on utilise la fonction `range`

```
for i in range(5):  
    print(i)
```

Le résultat est

```
0  
1  
2  
3  
4
```

# Python

**On peut aussi modifier la valeur initiale (par défaut 0)**

```
for i in range(2, 5):  
    print(i)
```



# Python

**On peut aussi modifier la valeur initiale (par défaut 0)**

```
for i in range(2, 5):  
    print(i)
```

**Le résultat est**

```
2  
3  
4
```

# Python

**Il est possible de modifier le pas (par défaut 1)**

```
for i in range(0, 5, 2):  
    print(i)
```

# Python

**Il est possible de modifier le pas (par défaut 1)**

```
for i in range(0, 5, 2):  
    print(i)
```

**Le résultat est**

```
0  
2  
4
```

# Python

Comme pour `while`, il est possible d'utiliser

- `pass`
- `continue`
- `break`
- `else`

# Python

**Il est possible de définir une boucle avec un pas négatif**

```
for i in range(5, 0, -1):  
    print(i)
```

# Python

**Il est possible de définir une boucle avec un pas négatif**

```
for i in range(5, 0, -1):  
    print(i)
```

**Le résultat est**

```
5  
4  
3  
2  
1
```

# Python

## Exercice 1

Écrire un code programme qui

- demande à l'utilisateur de saisir deux entiers  $a$  et  $b$  (avec  $a < b$ ),
- afficher les nombres pairs compris entre  $a$  et  $b$ .

# Python

## Exercice 2

Écrire un programme qui demande à l'utilisateur de saisir un entier positif  $n$  et qui calcule puis affiche la somme de tous les entiers positifs inférieurs à  $n$ .



# Python

Étant donnée la chaîne de caractères suivante

```
ma_chaine = "Bonjour tout le monde. On est dans le  
sud. Et il fait bon.";
```

# Python

Étant donnée la chaîne de caractères suivante

```
ma_chaine = "Bonjour tout le monde. On est dans le  
sud. Et il fait bon.";
```

## Exercice 3

Écrire un script **Python** qui permet de compter le nombre de phrases et celui de mots de la chaîne de caractères `ma_chaine`.

# Python

**Python nous permet de stocker le résultat d'une boucle `for` dans une variable (for-one-line)**

```
carre = [elt ** 2 for elt in range(5)]
```

```
print(carre)
```

```
# affiche [0, 1, 4, 9, 16]
```

# Python

Python nous permet de stocker le résultat d'une boucle `for` dans une variable (`for-one-line`)

```
carre = [elt ** 2 for elt in range(5)]
```

```
print(carre)
```

```
# affiche [0, 1, 4, 9, 16]
```

Dans un `for-one-line`, on peut utiliser un `if`

```
carre = [elt ** 2 for elt in range(5) if elt % 2 == 0]
```

```
print(carre)
```

```
# affiche [0, 4, 16]
```

# Python

On peut aussi imbriquer les `for`

```
voitures = ["peugeot", "fiat"]  
  
lettres = [x * 2 for elt in voitures for x in elt]  
  
print(lettres)  
  
# affiche ['pp', 'ee', 'uu', 'gg', 'ee', 'oo', 'tt',  
          'ff', 'ii', 'aa', 'tt']
```

# Python

## Exercice

Écrire un programme qui demande à l'utilisateur de saisir deux entiers positifs et qui calcule puis affiche leur plus grand diviseur commun.

# Python

## Exercice

- Étant donnée la variable suivante :  
`valeurs = [elt ** 2 for elt in range(5)]`
- En utilisant une boucle `for`
- Déterminez si cette variable contient un nombre pair

# Python

## Solution

```
valeurs = [elt ** 2 for elt in range(5)]
trouve = False

for elt in valeurs:
    if elt % 2 == 0:
        trouve = True
        break

print("oui" if trouve else "non")
```



# Python

**Le code précédent peut être simplifié en supprimant la variable `trouve` et en éliminant la deuxième structure conditionnelle**

```
valeurs = [elt ** 2 for elt in range(5)]

for elt in valeurs:
    if elt % 2 == 0:
        print("oui")
        break
    else:
        print("non")
```

# Python

```
for ... break ... else
```

- Si le compilateur rencontre un `break` dans la boucle `for`, le `else` ne sera pas exécuté.
- S'il ne rencontre pas `break` dans la boucle `for`, la partie `else` sera exécutée.

# Python

Pour indiquer à Python qu'une variable est présente, mais que sa valeur n'est pas utilisée à l'intérieur de la boucle, on utilise `_`

```
for _ in range(4):  
    print("bonjour")  
  
# affiche bonjour trois fois
```

# Python

Si une variable est définie dans un bloc `if` ou `for` au niveau le plus haut d'un script ou d'un module (c'est-à-dire pas à l'intérieur d'une fonction), alors elle est globale.

```
x = 2
if x > 0:
    resultat = f'{x} est positif'

print(resultat)
# affiche 2 est positif
```

# Python

## Constantes

- Certains langages, comme **Java**, **PHP** et d'autres permettent de définir des constantes.
- Une constante est une structure de données qui, une fois initialisée, ne peut plus changer de valeur.
- En **Python**, on n'a pas de mot-dé spécifique pour déclarer une constante comme dans certains autres langages de programmation.
- Cependant, la convention en **Python** est d'écrire le nom de la constante en majuscules pour indiquer qu'il s'agit d'une valeur constante.