

# Cours de Programmation en Python

## 2 Structures conditionnelles et boucles

# Structures conditionnelles

```
if condition:  
    bloc d'instructions  
else:  
    bloc d'instructions
```

Remarques:

- (1) Attention au : qui est primordial
- (2) C'est l'indentation (le décalage par rapport à la marge gauche) qui délimite le bloc d'instructions
- (3) La partie else est facultative

# Structures conditionnelles

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
else:
    print("a and b are equal")
```

```
a = 33
b = 200
if b > a:
    print("b is greater than a") # you will get an error
```

Noter l'imbrication des blocs.

Le code appartenant au même bloc doit être impérativement aligné sinon erreur.

# Structures conditionnelles

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

- elif n'est déclenché que si la (les) condition(s) précédente(s) a (ont) échoué.
- elif est situé au même niveau que if et else
- On peut en mettre autant que l'on veut

# Structures conditionnelles

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
a = 33
b = 200
if b > a:
    print("b is greater than a") # you will get an error
```

Noter l'imbrication des blocs.

Le code appartenant au même bloc doit être impérativement aligné sinon erreur.

# Condition et operateur Or And

Le mot-clé and est un opérateur logique et est utilisé pour combiner des instructions conditionnelles :

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

Le mot-clé or est un opérateur logique et est utilisé pour combiner des instructions conditionnelles :

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

# Boucles

Principe de la boucle for Elle ne s'applique que sur une collection de valeurs. Ex. tuples, listes,... ( suite du cours )

```
for indice in séquence:  
    bloc d'instructions
```

Remarque :

- Attention à l'indentation toujours
- On peut « casser » la boucle avec break ( mauvaise pratique )
- On peut passer directement à l'itération suivante avec continue( mauvaise pratique )
- Des boucles imbriquées sont possibles
- Le bloc d'instructions peut contenir des conditions

# Range

- (1) `range(4)` → 0 1 2 3
- (2) `range(1,4)` → 1 2 3
- (3) `range(0,5,2)` → 0 2 4

```
for x in range(6):  
    print(x)  
  
for x in range(2, 6):  
    print(x)
```



# Boucle while

```
while condition:  
    bloc d'instructions
```

Dans la condition : très important sur la condition , ça doit être une opération de comparaison

Attention à la boucle infinie !

```
i = 1  
while i < 6:  
    print(i)  
    i += 1
```

```
1  
2  
3  
4  
5
```

# Etapes de conception d'un programme

1. Déterminer les besoins et fixer les objectifs : que doit faire le logiciel, dans quel cadre va-t-il servir, quels seront les utilisateurs types ? On rédige un cahier des charges avec les stakeholder
2. Conception et spécifications : quels sont les fonctionnalités du logiciel, avec quelle interface ?
3. Programmation : modélisation et codage
4. Tests : obtient-on les résultats attendus, les calculs sont corrects, y a-t-il plantage et dans quelles circonstances ? (tests unitaires, tests d'intégration, etc.)
5. Déploiement : installer le chez le client (vérification des configurations, installation de l'exécutable et des fichiers annexes, etc.)
6. Maintenance : corrective, traquer les bugs et les corriger (patches) ; évolutive (ajouter des fonctionnalités nouvelles au logiciel : soit sur l'ergonomie, soit en ajoutant de nouvelles procédures)

# Specifications du langage Python

Python est un langage de programmation interprété. Il est associé à un interpréteur de commandes disponible pour différents OS (Windows, Linux, Mac OS X, etc.)

C'est langage complet qui dispose de : types de données, branchements conditionnels, boucles, organisation du code en procédures et fonctions, objets et classes, découpage en modules.

Très bien structuré, facile à appréhender, c'est un langage privilégié pour L'enseignement .

Comment exécuter ? transmettre à l'interpréteur Python le fichier script « .py »



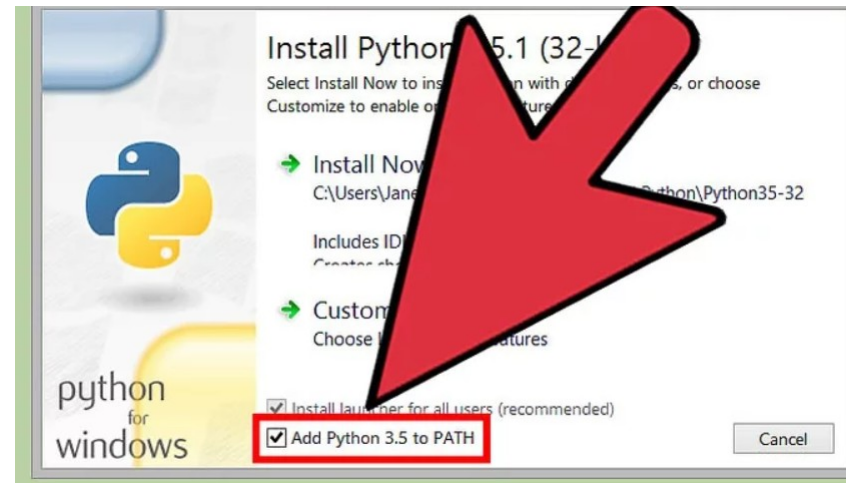
# Specifications du langage Python

- Données typées. Python propose les types usuels de la programmation : entier, réels, booléens, chaîne de caractères.
- Structures avancées de données: collections de valeurs (énumérations, listes) et des objets structurés (dictionnaires, classes)
- Séquences d'instructions, c'est la base même de la programmation, pouvoir écrire et exécuter une série de commandes sans avoir à intervenir entre les instructions.
- Structures algorithmiques : les conditions (structure conditionnelles) et les boucles.
- Possibilité de créer des procédures et fonctions
- Organisation du code en plusieurs modules réutilisable dans l'ensemble de vos programmes
- Python est sensible à la casse : il différencie les termes écrits en minuscule et majuscule.



# Installer Python

Telecharger python depuis le site <https://www.python.org/downloads/>



Telecharger et installer Anaconda ( solution recommandée)

# Installer Python ( Linux )

Vérifier si python est déjà installé

```
python3 --version
```

Installer a l'aide de apt

```
sudo apt update
```

```
sudo apt install python3
```

# Premiere utilisation de python

Exercice :

Ecriture du code dans un éditeur de code puis l'enregistrer dans un fichier « .py »  
`print("Hello World")`

Double cliquer le fichier « .py » pour lancer automatiquement le programme dans la console.



# Premiere utilisation de python

Créer un nouveau projet jupyter  
Selectionner une cellule et coller votre code  
Executer la cellule





# Types de variables

Le type d'une variable correspond à la nature de celle-ci. Les trois principaux types dont nous aurons besoin dans un premier temps sont :

- les entiers (integer ou int)

- les nombres décimaux que nous appellerons floats ou double

- les chaînes de caractères (string ou str). Bien sûr, il existe de nombreux autres types (par exemple, les booléens, les nombres complexes, etc.)

Declaration d'une variable , affectation et typage dynamique :

=> `a = 1.2`

`a` est une variable, elle a été automatiquement typée en decimal ( flottant ou « float » ) parce qu'il y a un point décimal. `a` est l'identifiant de la variable (attention à ne pas utiliser les mots réservés comme identifiant),

`=` est l'opérateur d'affectation



# Types de variables

- Calcul

- `d = a + 3`

d sera un réel contenant la valeur 4.2

- Forcer le typage d'une variable (sert aussi pour le transtypage)

- `b = float(1)`

Même sans point décimal, b sera considéré comme float (b = 1, il aurait été int dans ce cas).

- Connaître le type d'un objet

- `type(nom_de_variable)`

Affiche le type interne d'une variable (ex. `type(a)` → `<class 'float'>`)

- Supprimer un objet de la mémoire (pas beaucoup utilisée )

- `del nom_de_variable`

où `nom_de_variable` représente le nom de l'objet à supprimer



# Types de variables

- Numérique qui peut être int (entier) ou float (double). Les opérateurs applicables sont : + , - , \* , / (division réelle), \*\* (puissance) , % (modulo) , // (division entière)
- bool correspond au type booléen, il prend deux valeurs possibles True et False ( vrai ou faux ) (respecter la casse). Les opérateurs sont not (négation), and (ET logique), or (OU logique)

ex. not(True) → False ; True and False → False ; etc.

- str désigner les chaînes de caractères. Une constante chaîne de caractère doit être délimitée par des guillemets (ou des quotes)

ex. a ← « tano » affecte la valeur « tano » à l'objet a qui devient donc une variable de type chaîne de caractères. Une chaîne de caractère se comporte comme un vecteur : len() pour connaître sa longueur, a[0] → « t », a[1:3] → « an », a[2:] → « no », etc.

- Remarque : pour connaître la classe d'un objet i.e. le type associé à un objet, on utilise la fonction type(nom\_objet)

ex. type(1.2) → renvoie la valeur 'float'

# Types de variables

Affectation simple

#typage automatique

`a = 1.0`

#typage explicite

`a = float(1)`

La seconde évite les ambiguïtés.

Affectations multiples

#même valeur pour plusieurs variables

`a = b = 2.5`

#affectations parallèles

`a, b = 2.5, 3.2`

# Types de variables

Opérations particulières:

```
a = 2
```

```
a = a + 1
```

```
a++
```

Conversion en numérique

```
a = "144" # a est de type chaîne caractère
```

```
b = float(a) # b est de type float
```

N.B. Si la conversion n'est pas possible ex. `float(« PythonTest »)`, Python renvoie une erreur

Conversion en logique

```
a = bool("TRUE") # a est de type bool est contient la valeur True
```

```
a = bool(1) # renvoie True également
```

Conversion en chaîne de caractères

```
a = str(18) # a est de type chaîne et contient « 18 »
```

# Types de variables

Les operateurs sont très utiles pour l'usage des branchements conditionels

Les opérateurs de comparaison servent à comparer des valeurs de même type et renvoient un résultat de type booléen.

Sous Python, ces opérateurs sont `<`, `<=`, `>`, `>=`, `!=`, `==`

ex. `a = (12 == 13)` # a est de type bool, il a la valeur False

Car 12 est different de 13

`b = (12 < 13)` # b est de type bool, il a la valeur True Car 12 est bien inferieur strictement a 13