

# **Chapitre 1: Les bases de Python**

## ● Introduction

### → Pourquoi Python?

- Python est un langage polyvalent
- Python est un langage puissant
- Python est facile à apprendre
- Python est riche en bibliothèques

### **Limites**

- Python n'est pas le meilleur langage pour les systèmes distribués et les systèmes en temps réel.

## ● Introduction

### → Installation sous Windows

## • Installation de Python

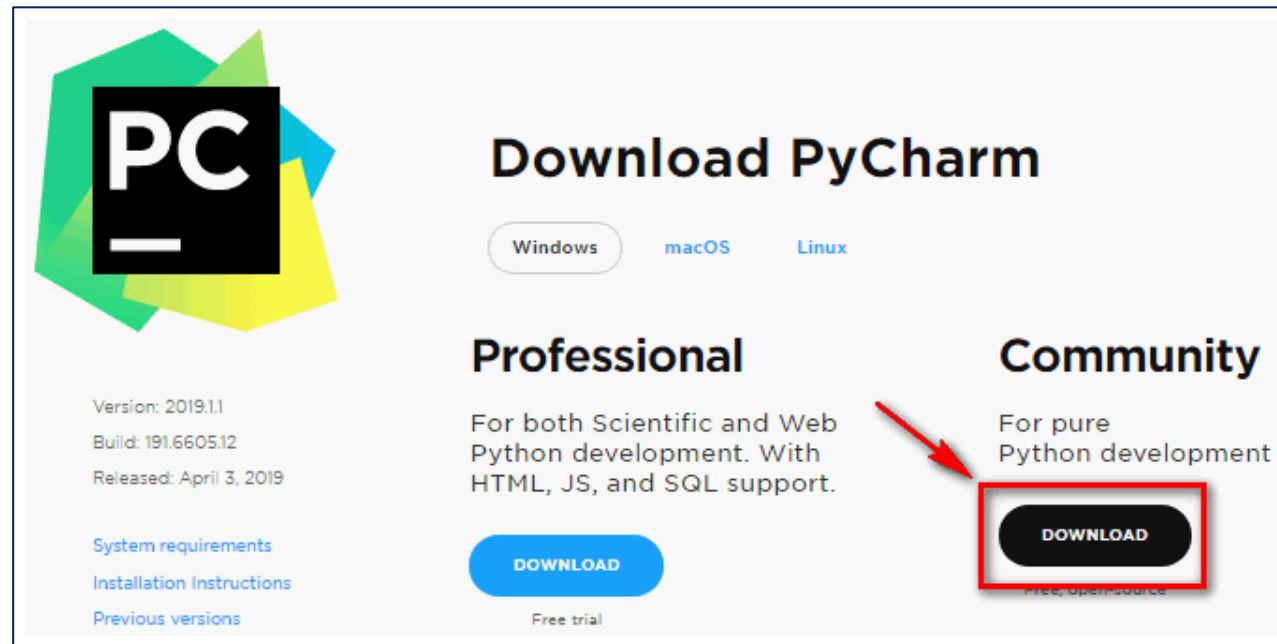
- Pour télécharger et installer Python, visitez le site officiel de Python <http://www.python.org/downloads/> et choisissez votre version.
- Procéder une installation normale comme la plupart des logiciels.



## ● Introduction

### → Installation sous Windows

- **Installation de Pycharm**
- Pour télécharger PyCharm, visitez le site Web <https://www.jetbrains.com/pycharm/download/> et cliquez sur le lien
- « DOWNLOAD » sous la section Communauté.



## ● Introduction

### → Installation sous UNIX

Déterminez si Python est déjà installé.

```
$ python --version
```

Sur les dérivés Debian, comme Ubuntu, utilisez APT :

```
$ sudo apt-get install python3.7
```

Sur Red Hat et dérivés, utilisez yum.

```
$ sudo yum install python37
```

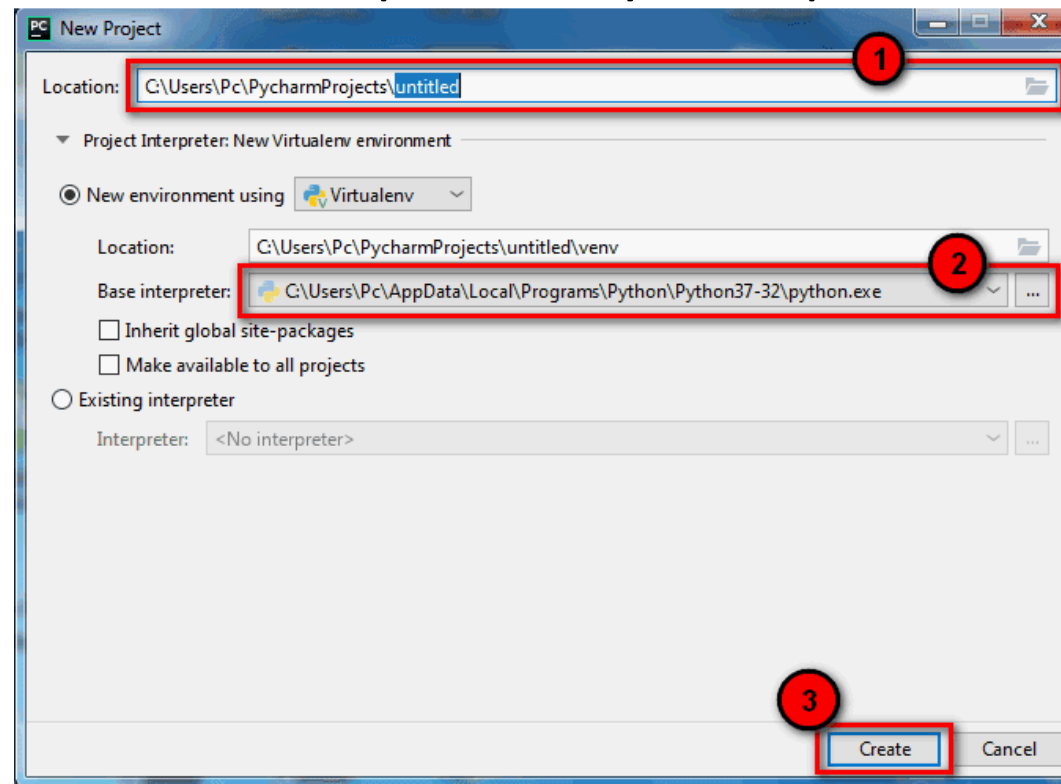
Sur SUSE et dérivés, utilisez zypper.

```
$ sudo zypper install python3-3.7
```

## ● Introduction

### → Création du premier programme

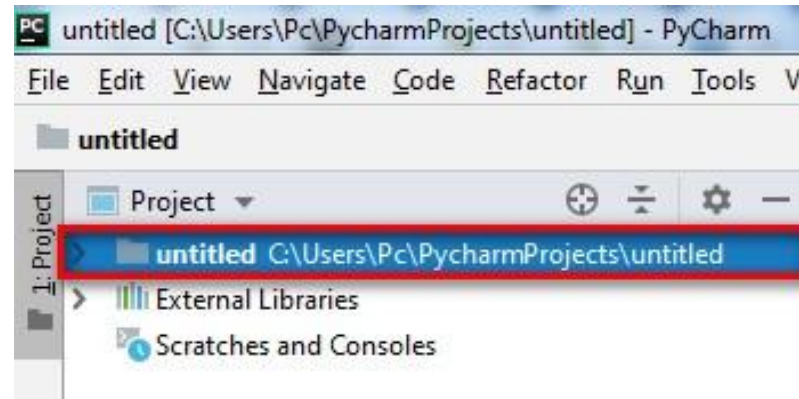
D'abord ouvrez l'éditeur PyCharm. Vous pouvez voir l'écran d'introduction à PyCharm. Pour créer un nouveau projet, cliquez sur "Create New Project". Vous pouvez sélectionner l'emplacement où vous souhaitez créer le projet. PyCharm aurait dû trouver l'interpréteur Python que vous avez installé précédemment.



## ● Introduction

### → Création du premier programme

Maintenant, sélectionnez le projet :



Ensuite, dans le menu « File », sélectionnez « New ». Ensuite, sélectionnez “Python File”.

Une nouvelle fenêtre apparaîtra. Maintenant, saisissez le nom du fichier que vous voulez (par exemple «HelloWorld») puis cliquez sur «OK».

Maintenant écrivez la ligne suivante – `print ('Hello World!')`.

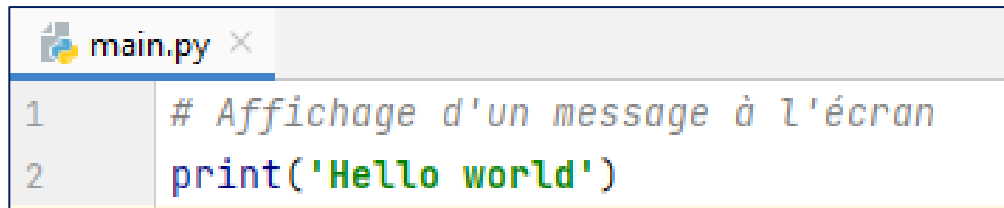
Ensuite, dans le menu « Run », sélectionnez « Run » pour exécuter votre programme.

## ● Introduction

### → Les commentaires en Python

Les commentaires commencent par un # et Python les ignorera:

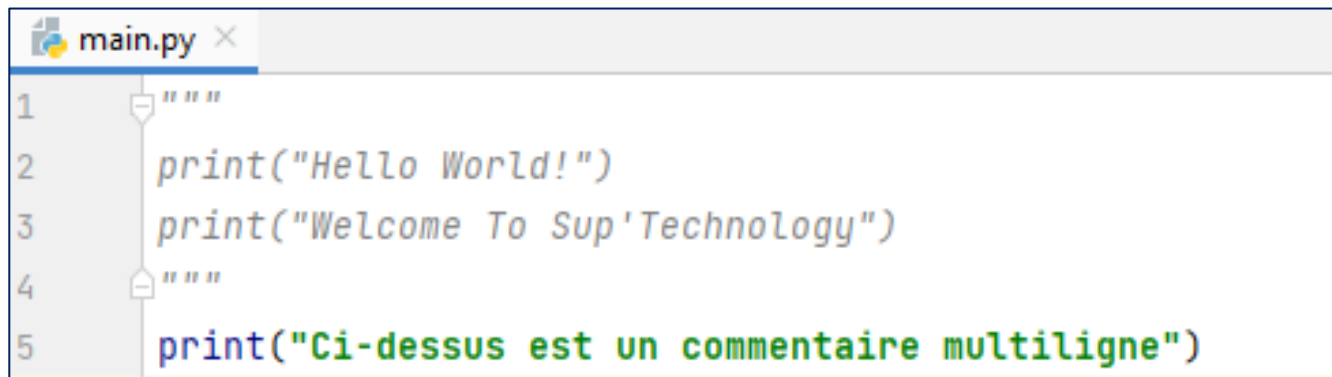
#### Exemple:



```
main.py x
1  # Affichage d'un message à l'écran
2  print('Hello world')
```

### Commentaires sur plusieurs lignes

Python n'a pas vraiment de syntaxe pour les commentaires sur plusieurs lignes. Pour ajouter un commentaire multiligne, vous pouvez insérer un # pour chaque ligne:



```
main.py x
1  """
2      print("Hello World!")
3      print("Welcome To Sup'Technology")
4  """
5  print("Ci-dessus est un commentaire multiligne")
```



## ● Les variables

### → Les nombres

Python prend en charge deux types de nombres: les nombres entiers et les nombres à virgule flottante.

```
main.py ×  
1 myint = 5  
2 print(myint)
```

Pour définir un nombre à virgule flottante, vous pouvez utiliser l'une des notations suivantes:

```
main.py ×  
1 myfloat = 5.0  
2 print(myfloat)
```

OU

```
main.py ×  
1 myfloat = float(5)  
2 print(myfloat)
```

Les affectations peuvent être effectuées sur plusieurs variables « simultanément » sur la même ligne comme celle-ci:

```
a, b = 5, 6  
print(a, b)
```

## ● Les variables

### → Les Strings

Les Strings ou les chaînes de caractères sont définies soit avec un guillemet simple ou un guillemets doubles.

```
mystring = 'Sup\'tech'  
print(mystring)
```

La concaténation ne se fait qu'entre les strings. Le code suivant génère une erreur:

```
age = 35  
mystring = "Mon âge est: " + age  
print(mystring)
```

Pour corriger le problème, il faut convertir en String comme suit:

```
age = 35  
mystring = "Mon âge est: " + str(age)  
print(mystring)
```

## ● Les variables

### → Conversion de type – Transtypage en Python

La conversion de type en python se fait à l'aide des fonctions constructeurs:

**int()** : Construit un nombre entier à partir d'un entier, d'un flottant (en arrondissant au nombre entier) ou d'une chaîne.

```
a = int(5)      # a sera 5
b = int(5.5)    # b sera 5
c = int("5")    # c sera 5
```

**float()** : Construit un nombre flottant à partir d'un entier, d'un flottant ou d'une chaîne (à condition que la chaîne représente un flottant ou un entier)

```
a = float(5)      # a sera 5.0
b = float(5.5)    # b sera 5.5
c = float("5")    # c sera 5.0
d = float("5.5")  # d sera 5.5
```

**str()** : Construit une chaîne à partir d'une grande variété de types de données, y compris des chaînes, des entiers et des flottants

```
a = str("ABC")    # a sera 'ABC'
b = str(5)         # b sera '5'
c = str(5.0)       # c sera '5.0'
```

## ● Les variables

### → supprimer une variable | Python

Python stocke les variables en mémoire après leur création. Python supprime automatiquement les variables et les fonctions lorsqu'elles ne peuvent plus être utilisées, libérant ainsi la mémoire utilisée. L'utilisateur peut également supprimer manuellement des variables. Cela peut être utile lorsque de grandes structures de données ne sont plus nécessaires, car leur suppression libérera de la mémoire pour d'autres utilisations.

```
1  x = 18
2  print(x)
3  del x
4  print(x)    ## renvoie une erreur ici car elle a déjà été supprimée.
```

## ● Les opérateurs en Python

### → Opérateurs arithmétiques Python

Les opérateurs arithmétiques Python sont utilisés avec des valeurs numériques pour effectuer des opérations arithmétiques courantes, telles que l'addition, la soustraction, la multiplication, la division, etc.

Opérateur	Nom	Exemple	Résultat
+	Addition	$a + b$	Somme de a et b
*	Multiplication	$a * b$	Produit de a et b
-	Soustraction	$a - b$	Différence de a et b
/	Division	$a / b$	Quotient de a et b
//	division entière	$a // b$	Résultat entier d'une division
%	Modulo	$a \% b$	Reste de a divisé par b
**	Exposant	$a ** b$	a à la puissance b

## ● Les opérateurs en Python

### → Opérateurs d'affectation Python

Les opérateurs d'affectation Python sont utilisés avec des valeurs numériques pour écrire une valeur dans une variable

Opérateur	Description	Exemple	Est la même que
=	Affectation simple	a = b	
+=	Addition puis affectation	a += b	a = a + b
-=	Soustraction puis affectation	a -= b	a = a - b
*=	Multiplication puis affectation	a *= b	a = a * b
/=	Division puis affectation	a /= b	a = a / b
%=	Modulo puis affectation	a %= b	a = a % b
//=	Division entière puis affectation	a //= b	a = a // b
**=	Exponentiation puis affectation	a **= b	a = a ** b
&=	Et bit à bit puis affectation	a &= b	a = a & b
=	Ou bit à bit puis affectation	a  = b	a = a   b
^=	XOR puis affectation	a ^= b	a = a ^ b
>>=	Décalage binaire à droite puis affectation	a >>= b	a = a >> b
<<=	Décalage binaire à gauche puis affectation	a <<= b	a = a << b

## ● Les opérateurs en Python

### ➔ Opérateurs de comparaison Python

Les opérateurs de comparaison Python sont utilisés pour comparer deux valeurs (nombre ou chaîne de caractères) :

Opérateur	Nom	Exemple	Résultat
==	Égal à	a == b	Retourne True si a est égal à b
!=	Non égal à	a != b	Retourne True si a n'est pas égal à b
>	Supérieur à	a > b	Retourne True si a est supérieur à b
<	Inférieur à	a < b	Retourne True si a est inférieur à b
>=	Supérieur ou égal à	a >= b	Retourne True si a est supérieur ou égal à b
<=	Inférieur ou égal à	a <= b	Retourne True si a est inférieur ou égal à b

## ● Les opérateurs en Python

### → Opérateurs logiques Python

Les opérateurs logiques Python sont utilisés pour combiner des instructions conditionnelles.

Opérateur	Exemple	Résultat
and	a and b	True si les deux a et b sont Trues
or	a or b	True si a ou b est True
not	!a	True si a n'est pas True

### → Opérateurs d'identité Python

Les opérateurs d'identité sont utilisés pour comparer les objets, non pas s'ils sont égaux, mais s'ils sont en fait le même objet, avec le même emplacement mémoire:

Opérateur	Description	Exemple
is	Renvoie True si les deux variables sont le même objet	a is b
is not	Renvoie True si les deux variables ne sont pas le même objet	a is not b



## ● Les opérateurs en Python

### ➔ Opérateurs d'appartenance Python

Les opérateurs d'appartenance sont utilisés pour tester si une séquence est présentée dans un objet:

Opérateur	Description	Exemple
in	Renvoie True si une séquence avec la valeur spécifiée est présente dans l'objet	a in b
not in	Renvoie True si une séquence avec la valeur spécifiée n'est pas présente dans l'objet	a not in b

## ● Les opérateurs en Python

### → Opérateurs binaires en Python

Les opérateurs binaires en Python sont utilisés pour comparer les nombres (binaires):

Opérateur	Nom	Exemple	Résultat
&	AND	a & b	Met chaque bit à 1 si les deux bits sont à 1
	OR	a   b	Définit chaque bit sur 1 si l'un des deux bits vaut 1
^	XOR	a ^ b	Définit chaque bit sur 1 si un seul des deux bits vaut 1
!=	Inégalité	a != b	Retourne True si a n'est pas égal à b
~	NOT	a ~ b	Inverse tous les bits
<<	Décalage vers la gauche	a << b	Décalage vers la gauche
>>	Décalage vers la droite	a >> b	Décalage vers la droite

## ● Fonctions d'entrée/sortie

### → Entrées clavier : `input()`

L'instruction `input()` permet de saisir une entrée au clavier

Elle effectue un typage dynamique (détecte le type de la valeur entrée)

Accepter une syntaxe `input(message)`

### Exemple

```
>>>n=input("Entrez un entier") # la valeur tapée sera affectée à n
8
>>> type(n)
<type 'str'>
```

Pour lire une variable d'un autre type, il faut faire un transtypage(cast)

```
>>> b=int(input())
5
>>> type(b)
<type 'int'>
```

## ● Fonctions d'entrée/sortie

### → Entrées clavier : `print()`

L'instruction `print()` permet d'afficher la valeur d'une variable à l'écran

### Exemple

```
>>> print ("salut") # En version 3.X c'est une fonction
Salut
>>> a= "IWA"
>>> print (a)
IWA
```

### Remarque:

- Elle fait un saut de ligne après l'affichage de la valeur
- Python 3.X: ajouter un paramètre `end=" "`

## ● Notion de bloc d'instruction

### → Définition et structuration

Ce type d'instruction permet au code de suivre différents chemins suivant les circonstances. Il s'agit, en quelque sorte, d'un aiguillage dans le programme.

Une instruction composée se compose :

- d'une ligne d'en-tête terminée par **deux-points** ;
- d'un bloc d'instructions indenté **par rapport à la ligne d'en-tête**. On utilise habituellement quatre espaces par indentation et on ne mélange pas les tabulations et les espaces.

Toutes les instructions au même niveau d'indentation appartiennent au même bloc

```
en-tete:  
    bloc .....  
    .....  
    d'instructions .....
```

## ● Notion de bloc d'instruction

### → Définition et structuration

Il est possible d'imbriquer des blocs d'instructions les uns dans les autres :

```
en-tete:
    bloc .....
    .....
    en-tete 2:
        bloc .....
        .....
        d' instructions .....
    d' instructions .....
    .....
```

### Remarque

Il ne faut JAMAIS mélanger les tabulations et les espaces au risque d'avoir l'erreur suivante:

**IndentationError: unindentdoesnot match anyouterindentation level**

Utilisé dans les boucles, conditions, fonctions, exceptions ...

## ● If...Else en Python

### → Syntaxe de l'instruction if en Python

```
if condition:  
    instruction(s)
```

En Python, le corps de l'instruction if est indiqué par l'indentation (espace au début d'une ligne). Le corps commence par une indentation et la première ligne non indentée marque la fin.

### Exemples:

```
a = 5  
b = 10  
if b > a:  
    print("B est supérieur à A")
```

```
print("A est supérieur à B") if a > b else print("A est inférieur ou égale à B")
```

## ● If...Else en Python

→ `if(...) elif ... else`

Les instructions Il est possible d'imbriquer plusieurs tests à l'aide du mot-clé `elif`

```
if expression booleenne1:
    bloc.....
    d instructions 1.. .....
elif expression booleenne2:
    bloc.....
    d instructions 2.....
else expression booleenne2:
    bloc.....
    d instructions 2. ....
```

- Si l'expression 1 est vraie, le bloc d'instructions 1 est réalisé ;
- Si l'expression 1 est fausse et l'expression 2 vraie, le bloc d'instructions 2 est réalisé ;
- Si les deux expressions sont fausses, le bloc d'instructions 3 est réalisé.

Plusieurs `elif` à la suite peuvent être utilisés pour multiplier les cas possibles.

`elif` est une concaténation de `else` et `if` et pourrait se traduire par « sinon si ».



## ● If...Else en Python

→ if(...) elif ... else

```
x = 5
if x < 0:
    print("x est négatif")
elif x % 2 != 0:
    print("x est positif et impair")
    print("ce qui est bien aussi !")
else:
    print("x n'est pas négatif et est pair")
```

## ● If...Else en Python

### → If imbriqué

Vous pouvez avoir des instructions if à l'intérieur des instructions if, cela s'appelle des instructions if imbriquées.

```
1  n = 10
2  if n >= 0:
3      if n == 0:
4          print("Zero")
5      else:
6          print("Nombre positif")
7  else:
8      print("Nombre négatif")
```

## ● If...Else en Python

### → Exercice

Écrivez un programme pour trouver un maximum entre trois nombres en utilisant une if-else ou if imbriquée.

```
num1 = int(input("Saisir le nombre 1 : "))
num2 = int(input("Saisir le nombre 2 : "))
num3 = int(input("Saisir le nombre 3 : "))

if num1 > num2:
    if num1 > num3:
        max = num1
    else:
        max = num3
else:
    if num2 > num3:
        max = num2
    else:
        max = num3

print("le maximum est = ", max)
```

## ● instructions répétitives

### → Définition

Ce type d'instruction permet au programme de répéter, de compter ou d'accumuler, avec une économie d'écriture considérable

On appelle boucle un système d'instructions qui permet de répéter un certain nombre de fois toute une série d'opérations.

Python propose deux instructions particulières pour construire des boucles :

- l'instruction **for ... in ...** , très puissante;
- l'instruction **while...** .

## ● instructions répétitives

### → La boucle « for »

L'instruction **for (... ) in** est une autre forme de boucle, qui permet cette fois d'itérer sur une collection de données, telle une liste ou un dictionnaire.

### Syntaxe

```
for élément in séquence
    bloc .....
    .....
    d instructions .....
```

- La séquence est parcourue élément par élément. L'élément peut être de tout type : entier, caractère, élément d'une liste...
- Il est possible d'itérer sur les indices d'une séquence (et non pas sur les éléments eux-mêmes) grâce à la fonction `range()` qui produit une séquence contenant les entiers de 0 à n-1.

## ● instructions répétitives

### → La boucle « for »

Afficher chaque nombre dans la liste des nombres:

```
nbrs = [1, 2, 3, 4, 5]
for n in nbrs:
    print(n)
```

## Parcourir une chaîne de caractère

Même les chaînes sont des objets itérables, elles contiennent une séquence de caractères. Exemple : Parcourez les lettres du mot «La formation IWA »:

```
for c in "La formation IWA":
    print(c)
```

## ● instructions répétitives

### → La boucle « for »

■ La fonction range peut prendre entre 1 et 3 arguments entiers :

- **range**(b) énumère les entiers 0,1,2, ..., b-1 ;
- **range**(a,b) énumère les entiers a,a+1,..., b-1 ;
- **range**(a,b,c) énumère les entiers a, a+c, a+2c, ..., a+nc où n est le plus grand entier vérifiant  $a + nc < b$  si c est positif (ie n est la partie entière de  $(b-a)/c$ ) et  $a + nc > b$  si c est négatif

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 15))
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> list(range(1, 20, 3))
[1, 4, 7, 10, 13, 16, 19]
```

- **instructions répétitives**

- La boucle « for »

- Exemples

```
for x in range(2, 10, 3):  
    print (x, x**2)
```

```
for x in range (1, 6):  
    for y in range (1, 6):  
        print (x * y, end=' ')  
    print('/')
```



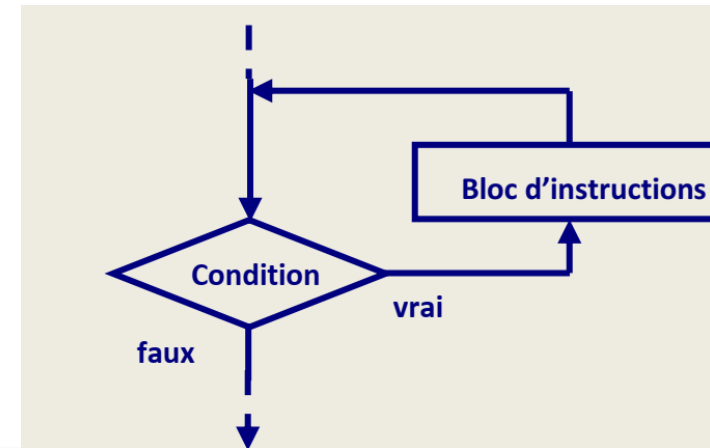
## ● instructions répétitives

### → La boucle « while »

Une boucle conditionnelle exécute une suite d'instructions tant qu'une certaine condition est réalisée.

#### Syntaxe

```
while condition:  
    bloc .....  
    D'instructions .....  
    à réaliser .....
```



On appelle boucle un système d'instructions qui permet de répéter un certain nombre de fois toute une série d'opérations.

## ● instructions répétitives

### → La boucle « while »

**Exemple:** La table de multiplication par 8 avec la boucle while

```
print("Table de multiplication par 8")

# initialisation de la variable de comptage
compteur = 1

while compteur <10 :

    # ce bloc est exécuté tant que la condition (compteur<10) est vraie
    print (compteur,"* 8 =",compteur*8)
    # incrémentation du compteur : compteur = compteur + 1
    compteur += 1

# on sort de la boucle
print("Eh voilà !")
```

## ● instructions répétitives

### → La boucle « while »

- Exemple: affichage de l'heure courante avec la boucle while

```
import time # importation du module time
quitter = 'n' # initialisation de la réponse
while quitter != 'o' :

    # ce bloc est exécuté tant que la condition (quitter != 'o') est
    # vraie
    print("Heure courante",time.strftime('%H:%M:%S'))
    quitter = input("Voulez-vous quitter le programme (o/n) ?")
# on sort de la boucle
print("A bientôt")
```

## ● instructions répétitives

### → Instruction break & continue

#### L'instruction break

Avec l'instruction break, nous pouvons arrêter la boucle avant d'avoir parcourir tous les éléments. Exemple : Quittez la boucle lorsque `str == "Python"`:

```
Lang = ["Java", "PHP", "Python", "C++", "Pascal"]
for str in lang:
    print(str)
    if str == "Python":
        break
```

#### L'instruction continue

Avec l'instruction continue, nous pouvons arrêter l'itération courante de la boucle for et continuer l'instruction suivante. Exemple: N'affichez pas « Python »:

```
Lang = ["Java", "PHP", "Python", "C++", "Pascal"]
for str in lang:
    if str == "Python":
        continue
    print(str)
```

## ● instructions répétitives

### → Exercice

Ecrire programme permettant de lire un nombre entier N puis calcule son factoriel.

- $N != 1 * 2 * 3 * \dots * (n-1) * N$
- $0 != 1$

En utilisant les boucles suivantes

1.while

2.for

```
# boucle for
a = int(input("saisir un nombre "))
F = 1
for i in range(2, a + 1):
    F *= i
print("le factoriel de ", a, " est ", F)
```

```
# boucle while
a = int(input("saisir un nombre "))
F = 1
Cpt = 2
while Cpt <= a:
    F *= Cpt
    Cpt += 1
print("le factoriel de ", a, " est ", F)
```

## ● instructions répétitives

### → Exercice

Ecrire un programme qui permet de saisir un entier N et d'afficher s'il est premier ou non. Un nombre est dit premier s'il est divisible uniquement par 1 et par lui-même.

```
N=int(input("saisir un nombre - "))
etat=True
for i in range(2,N) :
    if N%i==0 :
        etat=False
        break
if etat==True :
    print(N," est un nombre premier")
else :
    print(N," n est pas premier")
```

## ● Chaines de caractères

### → Définition

Similaire aux chaines de caractères des autres langages (tableau de caractères)

Accès aux caractères par index, ie: pour une chaine s, s[i] retourne le ième-1 caractère

Les constantes sont écrites entre `"""`, ou entre `' '`

On peut écrire des `" "` dans les chaines délimitées par `' '` et des `' '` dans les chaines délimitées par `" "`

### **Exemple:**

- enjapS= " Je m'appelle Ahmed"
- S= ' le mot " Oni" signifie démon onais '

## ● Chaines de caractères

### → Opération sur les chaîne de caractères

Accès aux éléments: chaîne de caractères  $S$  de taille  $n$

- Le premier caractère est  $S[0]$  et le dernier  $S[n-1]$
- $S[i]$  retourne le  $i$ ème -1 élément de  $S$
- $S[-i]$  retourne le  $i$ ème élément de  $S$  en partant de la fin. (à la position  $n-i$ )

### Sous chaînes

- $S[i:j]$  donne la sous-chaîne composée des caractères aux positions  $i, i+1, \dots, j-1$
- $S[:j]$  donne la sous-chaîne composée des caractères aux positions  $0, 1, \dots, j-1$
- $S[j:]$  donne la sous-chaîne composée des caractères aux positions  $j, j+1, \dots, n-1$



## ● Chaines de caractères

### → Opération sur les chaîne de caractères

**Exemple:** la longueur de la chaîne

```
s = "abcde"  
len(s)  
print(s)
```

**Exemple:** Répétition

```
s = "Fi! "  
s = s4 * 3  
print(s)
```

## ● Chaines de caractères

### → Fonctions

- **s.isalnum()**: retourne vrai si tous les caractères sont alphanumériques
- **s.isalpha()**: retourne vrai si tous les caractères sont alphabétiques
- **s.isdigit()**: retourne vrai si tous les caractères sont des chiffres
- **s.replace(old, new)**: retourne une chaîne où toutes les occurrences de **old** dans **s** sont remplacées par **new**
- **s.index(c)** : retourne la position de **c** dans **s**, -1 s'il n'y existe pas

## ● **Formatage de chaîne en Python**

Python utilise le style du langage C pour créer de nouvelles chaînes formatées. L'opérateur « % » est utilisé pour formater un ensemble de variables contenues dans un « tuple, ainsi qu'une chaîne qui contient du texte normal avec des symboles spéciaux comme « %s » et « %d ».

Disons que vous avez une variable appelée « name » et que vous souhaitez ensuite afficher un message.

```
name = "Thomas"  
print("Hello %s, Welcome to  
IWA!" % name)
```

Si vous avez deux ou plusieurs arguments, utilisez un tuple (parenthèses):

```
name = "Mohamed"  
age = 28  
print("My name is %s, I am %d years old." % (name, age))
```

## ● **Formatage de chaîne en Python**

Tout objet qui n'est pas une chaîne peut également être formaté à l'aide de l'opérateur %s. Par exemple:

```
dict = {1:"A", 2:"B", 3:"C"}  
print("My dictionnary: %s" % dict)
```

Voici quelques opérateurs de base que vous devez connaître:

%s – Chaîne (ou tout objet avec une représentation sous forme de chaîne)

%d – Entiers

%f – Nombres à virgule flottante

Vous pouvez aussi formater une chaîne en utilisant la méthode format().

## ● **Formatage de chaîne en Python**

### → **La méthode format()**

La méthode format() vous permet de formater des parties sélectionnées d'une chaîne. Parfois, il y a des parties d'un texte que vous ne contrôlez pas, peut-être, elles proviennent d'une base de données ou une entrée d'un utilisateur.

Pour contrôler ces valeurs, ajoutez des accolades {} dans le texte et exécutez les valeurs via la méthode format():

```
name = "Mohamed"  
str = "Hello {}, Welcome to SupTech!"  
print(str.format(name))
```

Si vous souhaitez utiliser plus de valeurs, ajoutez simplement plus de valeurs à la méthode format():

```
name = "Mohamed"  
age = 18  
str = "My name is {}, I am {} years old."  
print(str.format(name, age))
```

## ● **Formatage de chaîne en Python**

### → **Numéros d'index**

Vous pouvez utiliser des numéros d'index (un nombre entre accolades {0}) pour vous assurer que les valeurs sont placées dans les espaces appropriés.

```
num = 4
rue = 63
quartier = 90
adresse = "N°{0}, Rue {1} Quartier {2} Casablanca."
print(adresse.format(num, rue, quartier))
```

Ensuite, si vous souhaitez faire référence à la même valeur plusieurs fois, utilisez le même numéro d'index.

```
heure = 9
min = 30
ladate = "Du {0}h{1}min du matin, à {0}h{1}min du soir."
print(ladate.format(heure,min))
```

## ● Les Fonctions en Python

### → Syntaxe

Python fournit des fonctions intégrées telles que `print()`, `input()` etc., mais nous pouvons également créer vos propres fonctions. Ces fonctions sont appelées « des fonctions définies par l'utilisateur ».

### Syntaxe d'une fonction

```
def functionName (arguments) :  
    //instruction 1  
    //instruction 2  
    //...  
    return [expression]
```

### Exemple :

```
def sayHello (name) :  
    print("Hi, " + name + " !")  
sayHello("Mohamed")
```

## ● Les Fonctions en Python

### → Arguments de fonction en Python

#### Arguments par défaut:

Les valeurs par défaut indiquent que l'argument de la fonction prendra cette valeur si aucune valeur d'argument n'est transmise lors de l'appel de la fonction.

```
def my_function(nbr1, nbr2 = 10):
```

#### Nombre variable d'arguments:

Ceci est très utile lorsque nous ne savons pas le nombre exact d'arguments qui seront passés à une fonction.

#### Définition de fonction:

```
def my_function(*varargs):
```

#### Appel de fonction:

```
my_function(10, 20, 30, 40)
```



## ● Les Fonctions en Python

### → L'instruction return

#### L'instruction return

L'instruction return permet de quitter une fonction et renvoie éventuellement un résultat à l'appelant. Une instruction return sans argument est identique à « return None ».

```
def somme(nbr1, nbr2):  
    return nbr1 + nbr2  
  
print(somme(10, 20))
```

## ● Les Fonctions en Python

### → \*args et \*\*kwargs en Python

#### **\*args**

\*args est utilisé pour envoyer une liste d'arguments de longueur variable sans mot-clé à la fonction.

```
# *args pour un nombre variable d'arguments  
def display(*argv) :  
    for arg in argv:  
        print(arg)  
  
display('Hello', 'Welcome', 'to', IWA)
```

## ● Les Fonctions en Python

### → \*args et \*\*kwargs en Python

#### **\*\*kwargs**

\*\*kwargs vous permet de transmettre une liste d'arguments de longueur variable avec des mots-clés à une fonction. Vous devez utiliser \*\*kwargs si vous souhaitez gérer des arguments nommés dans une fonction.

```
def display(**kwargs):  
    for key, value in kwargs.items():  
        print("%s == %s" % (key, value))  
  
display(id=1, name='Mohamed', age=23)
```

## ● Les Fonctions en Python

### → Exercice

Écrire un programme qui permet de saisir deux entiers  $n$  et  $p$  (tq  $1 < p < n$ ) et de calculer puis afficher le nombre de combinaison de  $p$  éléments parmi  $n$  : CNP

sachant que

$$C_n^p = \frac{n!}{p! * (n - p)!}$$

Pour résoudre ce problème on besoin de définir les fonctions suivants:

- **Saisir( )** une fonction qui permet de saisir et retourner  $n$  et  $p$ .
- **Fact( n)** une fonction permet de calculer et retourner la factorielle de  $n$ .
- **Calculer( n, p)** une fonction permet de calculer la CNP.

## ● Les variables

### → Variable locale et globale | Python

#### Variables locales:

Les variables locales ne peuvent être atteintes que dans leur portée. L'exemple ci-contre a deux variables locales: a et b. Les variables a et b ne peuvent être utilisées qu'à l'intérieur de la fonction 'add', elles n'existent pas en dehors de la fonction.

```
def add(a, b):  
    res = a + b  
    return res  
print(add(1, 3))
```

#### Variables globales:

Une variable globale peut être utilisée n'importe où dans le code.

Dans l'exemple ci-dessous, nous définissons une variable globale b:

```
b = 5  
def show():  
    global b  
    b = 10  
show()  
print(b)
```

## ● Les variables

### → Variable locale et globale | Python

#### Exercice

Déterminer la sortie de ce programme:

```
1  b = 10
2  def f1():
3      global b
4      b = 3
5  def f2(p,q):
6      global b
7      return p + q + b
8  f1()
9  res = f2(1,2)
10 print(res)
```

## ● Les modules en Python

### → Définition

Un module vous permet d'organiser logiquement votre code Python. Le regroupement du code associé dans un module facilite la compréhension et l'utilisation du code.

Simplement, un module est un fichier composé de code Python. Un module peut définir des fonctions, des classes et des variables

### → Créer un module:

Pour créer un module, enregistrez simplement le code que vous voulez dans un fichier avec l'extension .py.

### Exemple:

Enregistrez ce code dans un fichier nommé « my\_module.py ». Vous pouvez nommer le fichier de module comme vous le souhaitez, mais il doit avoir l'extension de fichier .py

```
def sayWelcome(name):  
    print(name + ", Welcome to IWA!")
```

## ● Les modules en Python

### → Utiliser un module

Maintenant, nous pouvons utiliser le module que nous venons de créer, en utilisant l'instruction `import`. L'exemple suivant importe le module nommé `my_module` et appelle la fonction `sayWelcome()`:

```
import my_module  
my_module.sayWelcome("Mohamed")
```

### → Variables dans un module

Un module peut contenir des fonctions, mais aussi des variables de tous types (listes, tuples, dictionnaires, objets, etc...).

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]
```

Importez le module nommé `my_module` et accédez à la liste:

```
import my_module  
color = my_module.liste[1]  
print(color)
```



## ● Les modules en Python

### → Renommer un module

Vous pouvez créer un alias lorsque vous importez un module, en utilisant le mot clé as:

```
import my_module as msg  
my_module.sayWelcome("Mohamed")
```

### → Importer à partir du module

Vous pouvez choisir d'importer uniquement les parties d'un module, en utilisant le mot-clé from.

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]  
def sayWelcome(name):  
    print(name + ", Welcome to IWA!")
```

L'exemple suivant importe uniquement la fonction sayWelcome() du module:

```
from my_module import sayWelcome  
sayWelcome("Mohamed")
```

## ● Les modules en Python

### ➔ Modules intégrés

Il existe plusieurs modules intégrés en Python, que vous pouvez importer à tout moment.

### Exemple

Importez et utilisez le module 'math':

```
import math  
x = math.sqrt(4)  
print(x)
```

## ● Les modules en Python

### → Modules intégrés

### Utilisation de la fonction `dir()`

Il existe une fonction intégrée pour connaître les fonctions (ou les variables) dans un module Python. La fonction `dir()`:

### Exemple

Lister tous les fonctions/variables définis dans le module 'math':

```
import math
x = dir(math)
print(x)
```

**Remarque:** La fonction `dir()` peut être utilisée sur tous les modules, même ceux que vous créez vous-même.

## ● Date et Heure en Python

Python possède un module nommé `datetime` pour travailler avec les dates et les heures. Regardons quelques exemples.

### Exemple 1: Récupérer la date et l'heure actuelles

```
import datetime
date = datetime.datetime.now()
print(date)
```

Le module 'datetime' possède de nombreuses méthodes pour renvoyer des informations sur l'objet `date`.

L'exemple suivant renvoie l'année et le nom du jour de la semaine:

```
import datetime
date = datetime.datetime.now()
print(date.year)
print(date.strftime("%A"))
```

## ● Date et Heure en Python

Python possède un module nommé `datetime` pour travailler avec les dates et les heures. Regardons quelques exemples.

### Exemple 1: Récupérer la date et actuelle

```
import datetime
date = datetime.datetime.today()
print(date) #Renvoie 2022-11-11
```

#### → Créer un objet Date

La classe `datetime()` nécessite trois paramètres pour créer une date: année, mois, jour. L'exemple suivant crée un objet Date:

```
import datetime
date = datetime.datetime(2021, 6, 21)
print(date) #Renvoie 2021-06-21 00:00:00
```

## ● Date et Heure en Python

### → Récupérer la date à partir d'un timestamp

Un timestamp Unix est le nombre de secondes entre une date particulière et le 1er janvier 1970 à UTC. Vous pouvez convertir un timestamp en date à l'aide de la méthode `fromtimestamp()`.

```
from datetime import date
timestamp = date.fromtimestamp(1623759849)
print("Date =", timestamp)
```

### → Créer un objet Time

```
from datetime import time
t1 = time()
print("t1 = ", t1)
t2 = time(12, 50, 59) # time(hour, minute and second)
print("t2 = ", t2)
```

## ● Date et Heure en Python

### → Afficher heure, minute, seconde et microseconde

Une fois que vous avez créé un objet time, vous pouvez facilement afficher ses attributs tels que l'heure, minute, seconde, etc.

```
from datetime import time
t = time(12, 50, 59)
print("heure = ", t.hour)
print("minute = ", t.minute)
print("seconde = ", t.second)
print("microseconde = ", t.microsecond)
```

## ● Date et Heure en Python

### → Afficher l'année, le mois, l'heure, la minute, et la seconde

Une fois que vous avez créé un objet time, vous pouvez facilement afficher ses attributs tels que l'heure, minute, seconde, etc.

```
from datetime import datetime
t = datetime(2021, 12, 30, 12, 55, 59)
print("année =", t.year)
print("mois =", t.month)
print("heure =", t.hour)
print("minute =", t.minute)
print("seconde =", t.second)
```