Chapitre 1 : Introduction générale sur le langage de programmation et Python

I. Un langage de programmation :

On peut dire que chaque langue (qu'elle soit mécanique ou naturelle, peu importe) est composée des éléments suivants :

UN ALPHABET :

Un ensemble de symboles utilisés pour construire les mots d'une certaine langue (par exemple, l'alphabet latin pour l'anglais, l'alphabet cyrillique pour le russe, le kanji pour le japonais, etc.).

UN LEXIS :

Chaque langage de programmation a son dictionnaire et vous devez le maîtriser

SYNTAXIQUEMENT :

Chaque langue a ses règles et celles-ci doivent être respectées

SEMANTIQUEMENT :

Le programme doit avoir un sens.

Malheureusement, un programmeur peut aussi faire des erreurs avec chacun des quatre sens mentionnés ci-dessus. Chacun d'entre eux peut rendre le programme complètement inutile.

Supposons que vous ayez écrit un programme avec succès. Comment persuader l'ordinateur de l'exécuter ? Vous devez traduire votre programme en langage machine. Heureusement, la traduction peut être effectuée par l'ordinateur lui-même, ce qui rend le processus rapide et efficace.

Il existe deux façons différentes de **transformer un programme d'un langage de programmation de haut niveau en langage machine : COMPILATION ET INTERPRETATION**

II. La Compilation

Le programme source est traduit une fois (cependant, cet acte doit être répété chaque fois que vous modifiez le code source) en obtenant un fichier (par exemple, un fichier .exe si le code est destiné à être exécuté sous MS Windows) contenant le code machine ; vous pouvez maintenant distribuer le fichier dans le monde entier ; le programme qui effectue cette traduction est appelé un compilateur ou un traducteur.

III. L'Interpretation

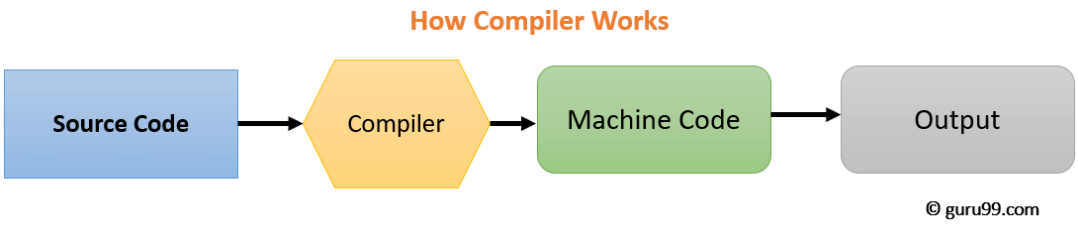
On peut traduire le programme source chaque fois qu'il doit être exécuté ; le programme effectuant ce type de transformation est appelé un interprète, car il interprète le code chaque fois qu'il doit être exécuté ; cela signifie également que vous ne pouvez pas simplement distribuer le code source tel quel, car l'utilisateur final a également besoin de l'interprète pour l'exécuter.

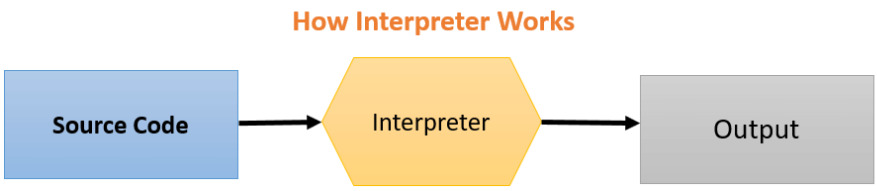
Exemple : Cas d'un film

- **Un compilateur** peut être comparé au fait de prendre un film étranger et de le sous-titrer d'abord, puis de pouvoir le regarder encore et encore.
- Tandis qu'un **interprète**, c'est comme si un traducteur traduisait chaque ligne du discours d'un délégué en temps réel.

IV. Compilation VS Interpretation

- Le compilateur traduit l'intégralité du programme avant son exécution.
- Les interprètes traduisent une ligne à la fois pendant l'exécution du programme.





V. Language de Programmation : PYTHON

Il existe plusieurs languages de Programmation notemment : C , C# , JAVA , PHP , COBOL , C++, ..et **PYTHON** . Chaque language de programmation a ces propres règles , un syntaxe , un lexique ..

➔ **C'est quoi PYTHON ?**

PYTHON est un langage de programmation de haut niveau, largement utilisé, c'est un language **interprété** et orienté objet, avec une sémantique dynamique, utilisé pour la programmation générale.

Sa première version est sorti en 1991 .

➔ **Qui a crée PYTHON ?**

En 1989, Guido Van Rossum, un développeur Néerlandais commence à travailler sur un nouveau langage de programmation en profitant d'une semaine de vacances. Il profite ensuite de son temps libre pour poursuivre le développement de son langage qu'il nomme Python en hommage à la série « Monty Python's Flying Circus ».

➔ **Les Objectifs de PYTHON**

En 1999, Guido van Rossum a défini ses objectifs pour Python :

- Un langage facile et intuitif tout aussi puissant que ceux des principaux concurrents.

- Une source ouverte, afin que chacun puisse contribuer à son développement.
- Un code aussi compréhensible qu'un simple anglais.
- Adapté aux tâches quotidiennes, permettant des temps de développement courts.

➔ **Les utilités de PYTHON**

Il est largement utilisé pour mettre en œuvre des **services Internet** complexes comme les moteurs de recherche, le stockage et les outils dans le nuage, les médias sociaux, etc. Chaque fois que vous utilisez l'un de ces services, vous êtes en fait très proche de Python.

De nombreux **outils de développement** sont implémentés en Python.

De plus en plus **d'applications d'usage quotidien** sont écrites en Python. Beaucoup de **scientifiques** ont abandonné les outils propriétaires coûteux pour passer à Python. De nombreux **testeurs** de projets informatiques ont commencé à utiliser Python pour effectuer des procédures de test répétables.

➔ **Les versions de PYTHON :**

PYTHON 2 ancienne , PYTHON 3 récente :

Python 3 est la version la plus récente (pour être précis, la version actuelle) du langage. Il suit son propre chemin d'évolution, créant ses propres normes et habitudes.

Le premier est plus traditionnel, plus conservateur que Python, et ressemble à certains des bons vieux langages dérivés du langage de programmation C classique.

VI. Environnement de Développement IDE :

Pour Développer avec Python il faut avoir un environnement de développement, Python a son propre IDE qui s'appelle : **IDLE** acronyme : « **I**ntegrated **D**evelopment and **L**earning **E**nvironment », ce qui veut dire un environnement de développement intégré de Python qui est une interface utilisateur permettant d'éditer votre code, l'exécuter et faire pleins de choses utiles que l'on verra par la suite. Cette application ressemble Dev C++ du langage C et C++, Visual Studio pour C#...

Chaque IDE Contient ces outils :

- Un **éditeur** qui vous aidera à écrire le code (il doit présenter certaines caractéristiques particulières, non disponibles dans les outils simples) ; cet éditeur dédié vous donnera plus que l'équipement standard du système d'exploitation (bloc-notes ou autres…).
- Une **console** dans laquelle vous pouvez lancer votre code nouvellement écrit et l'arrêter de force lorsqu'il devient incontrôlable.
- Un outil appelé **débogueur**, capable de lancer votre code étape par étape et vous permettant de l'inspecter à chaque instant de son exécution.

Pour Obtenir IDLE il suffit d'installer PYTHON 3 sur votre machine Pour le faire rendez vous sur le site officiel python et télécharger et installer python 3

Chapitre 2 : Fonction d'affichage : Print

I. La Fonction « Print » :

C'est une fonction **intégrée** très importante de Python, elle permet **d'afficher, au niveau de la console, les valeurs fournies en arguments** (càd entre parenthèses et séparés par des virgules « , »), et à l'affichage ces valeurs sont séparées par défaut par des **espaces** " " et à la fin, la fonction termine le travail par un **saut à la ligne**.

Syntaxe :

print(argument1,argument2)

Exemple :

Nous allons afficher des chaines de caractères pour le moment

```
print("Mon premier message")
```

Le code comprend les parties suivantes :

- Le nom de la fonction: **print**.
- Deux parenthèse (ouvrante, fermante)
- Deux guillemets ".." englobent notre message ou entre apostrophes'.. '
- La chaine de caractère : Mon premier message

➔ Ce ligne de code affiche le message *Mon premier message* et un retour a la ligne.

```
print("Je m'appelle Mohamed")
print("Mohamadi")
```

Le résultat est :

Je m'appelle Mohamed

Mohamadi

L'appel de la fonction sans Argument print() permet d'afficher une ligne vide.

Exemple :

```
print("Je m'appelle Mohamed")
print()
print("Mohamadi")
```

Le résultat est :

Je m'appelle Mohamed

Mohamadi

II. Les caractères d'échappement :

Le symbole **** est spécial : il permet de transformer le caractère suivant :

- **\n** est un saut de ligne
- **\t** est une tabulation
- **\'** est un « ' », mais il ne ferme pas la chaine de caractères
- **\"** est un « " », mais il ne ferme pas la chaine de caractères
- **** est un « \ »

Exemple :

```
print("Je m'appelle\n Mohamed")
print("Mohamadi")
```

Résultat est :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/te.py ====
Je m'appelle
 Mohamed
Mohamadi
>>>
```

III. Print avec plusieurs arguments :

Nous pouvons utiliser la fonction print avec plusieurs argument comme le montre l'exemple suivant :

```
print("Je","suis","belle")
```

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/te.py ====
Je suis belle
```

- Les arguments sont séparés par des virgules
- L'interpreteur ajoute un espace entre deux arguments

a- Argument Mot-clé :

Il existe deux a deux arguments de mots clés utiliser par la fonction print() : end et sep

Syntaxe:

argument mot clé=valeur affectée à cet argument

l'argument mot clé doit être placé après le dernier argument positionnel(c'est très important)

Argument mot clé end : l'argument de mot-clé end détermine les caractères que la fonction print() envoie à la sortie une fois qu'elle atteint la fin de ses arguments positionnels. Par défaut ce paramètre est égale a "\n".

Argument mot clé sep : Le paramètre sep spécifie le séparateur entre les arguments positionnel de la fonction print. Par défaut ce paramètre est égale a " ".

Exemple :

Code	Résultat
`print("Riyad","Zaid", end="")`	**Riyad Zaid**
`print("Riyad","Zaid", end="#")`	**Riyad Zaid#**
`print("Riyad","Zaid", end="/n")`	**Riyad Zaid**

`print("Riyad","Zaid", sep="-")`	**Riyad-Zaid**
`print("Riyad","Zaid", sep="")`	**RiyadZaid**
`print("Riyad","Zaid", sep="-", end="$")`	**Riyad-Zaid$**

IV. Exercices

Exercice 1 :

Ecrire un programme en python qui affiche à l'utilisateur les messages suivants :

Aujourd'hui il fait très beau

Tapez « + » si vous choisissez la somme :

Exercice 2 :

Ecrire un programme Python permettant d'afficher les textes suivants tels quels :

"I'm"

""learning""

"""Python"""

Exercice 3 :

Modifier le programme suivant :

```
print("Programming","Essentials","in")
print("Python")
```

Pour afficher le message suivant :

```
Programming***Essentials***in...Python
```

Chapitre 3 : Types Des Données / Les Variables

I. Les Types des Données :

Reflexion :

```
print("2")
print(2)
```

On constate qu'au niveau de l'affichage en console les 2 print() affichent le même résultat « 2 », mais en fait les manières de stocker en mémoire par la machine et manipuler ces 2 données sont complétement différentes.
Dans le 1er cas, il s'agit du caractère «2», tout comme les lettres « a » ou « z ».
Dans le second cas, il s'agit du nombre « 2 » qui a une toute autre signification.
Python est capable de reconnaitre chaque donnée entrée et de définir son type il existe 4 types de données

1- Les entiers (Integers) :

Chaque donnée numérique sans virgules on les appelle des entiers en anglais : **Integers** (**int** comme abréviation) qu'ils étaient positifs ou négatifs . Normalement on travaille avec des données numériques à la base décimal mais on peut tout de même représenter les données numériques dans 3 bases : Binaire / Octale / Hexadécimal

Représentation binaire (base 2) : Les nombres ne peuvent être représentés qu'avec les 2 chiffres 0 et 1, (rappelez-vous pour la base 10, on avait 10 chiffres : 0 à 9).

Exemple : Le nombre 1101 en binaire vaut : 1 x 20 + 0 x 21 + 1 x 22 + 1 x 23 = 13 (en base 10)

Représentation octale (base 8) : Les nombres ne peuvent être représentés qu'avec les 8 chiffres 0 à 7

Exemple : Le nombre 227 en octale vaut : 7 x 80 + 2 x 81 + 2 x 82 = 151 (en base 10)

Représentation hexadécimale (base 16) : Les nombres ne peuvent être représentés qu'avec les 16 symboles (0 à 9 ensuite A pour 10, B pour 11, C pour 12, D pour 13, E pour 14 et F pour 15), c'est tout à fait normal de ne représenter 10 à 15 par un seul symbole sinon ça représenterait d'autres nombres et ce serait faux !

Exemple : Le nombre A2C en hexadécimale vaut :

12 x 160 + 2 x 161 + 10 x 162 = 2604 (en base 10) 25

Le langage python reconnaît bien évidement ces 3 représentations, ainsi pour représenter un nombre en binaire, il suffit de le faire précéder par **0b (zéro b)**, par exemple : 0b1101

Pour représenter un nombre en octale, il faut le préfixer par **0o (Zéro o)**, par exemple : 0o227

Pour représenter un nombre en hexadécimale, il faut le préfixer par **0x (Zéro x)**, par exemple : 0xA2C

Voici ce que cela donne avec la fonction print() (Constatez que l'on retrouve les mêmes résultats que plus haut) :

```
print(0b1101)
```

```
Console >_
13
```

```
print(0o227)
```


```
151
```

```
print(0xA2C)
```


```
2604
```

2- Les réels ou flottants (Float) :

Comme le type entier « integer » qu'on vient de voir, il existe dans Python un autre type les nombres réels ou flottants en anglais « **float** » de nombre avec une partie fractionnaire (après la virgule) non vide.

Exemples : 2.5 ; -0.5 ; 256.1789

Important : le nombre 4 tout court est considéré comme entier par Python, en revanche 4.0 est considéré comme un float.

Il faut préciser qu'un « float » se note 2.5 (point) et non 2,5 (la virgule n'est pas admise, elle sert à autre chose dans Python !).

```
print(2.5)
```

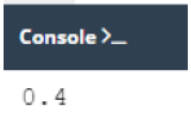

```
2.5
```

```
print(2,5)
```

Dans ce 2ème exemple la virgule « , » signifie que la fonction print() a 2 arguments qui sont les 2 nombres : 2 et 5, d'où l'affichage :

Pour Python le nombre décimal 0.4 peut être écrit .4 (en enlevant le zéro), mais ceci ne marche bien évidemment que pour le zéro :

```
print(.4)
```



De même le nombre 4.0 peut être écrit comme 4. (en enlevant le zéro) et cela ne changera ni son type (float) ni sa valeur :

```
print(4.)
```

Float VS Integer :

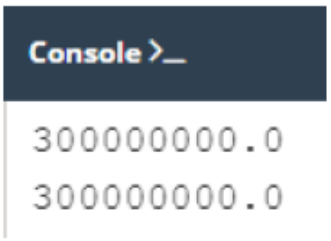
Un chose importante à savoir est que le point « . » du décimal est essentiel dans Python pour reconnaître les nombre à virgule flottante, en effet, le nombre 4 tout court est un entier (integer) pour Python, alors que 4.0 est un décimal (float) !

Un autre moyen d'utiliser les « float » est la lettre « e » comme exposant. Ce type de notation est particulièrement intéressant pour les très grands ou très petits nombres réels pour lesquelles la notation scientifique (avec exposant) s'avère intéressante.

Exemple : la vitesse de la lumière est de 300000000 m/s ou 3.108 m/s

En Python on l'écrit comme 3e8 ou 3E8 (il est nécessaire que l'exposant soit un nombre entier !)

```
print(3e8)
print(3E8)
```



3- Chaines de caractères (Strings)

Les chaînes de caractères sont aussi un type Python tout comme integer et float. On les utilise lorsqu'on doit traiter des textes tout simplement.

Par exemple le texte "Je suis une chaîne de caractères" est justement une chaine de caractères (string). Donc en Python tout ce qui est mis entre double quotes (guillemets) représente une chaîne de caractères.

Exemple : On souhaite afficher le texte tel quel avec les **" "** : **I like "Monty Python"**

On peut le faire de 2 manières différentes :

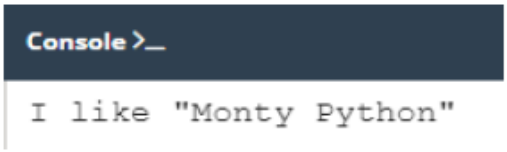
La 1ère consiste à utiliser le fameux caractère d'échappement « **** » avant chacun des guillemets :

```
print("I like \"Monty Python\"")
```

La 2ème solution consiste à utiliser l'apostrophe (simple quote) au lieu des doubles quotes pour spécifier une chaine de caractères. Simple quote ou double quote pour les string c'est pareil !

Mais si vous commencez une chaîne avec une apostrophe, vous devez la terminer avec une apostrophe, il faut être cohérent. Dans ce cas, pas besoin du caractère d'échappement « \ » :

```
print('I like "Monty Python"')
```



Remarque : Un chaine de caractères peut être vide, on la note : "" ou ''

4- Les données Booléenes :

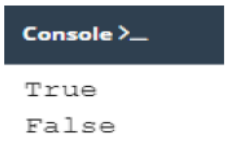
Les valeurs booléennes représentent un autre type Python comme ceux qu'on a vu jusqu'à maintenant mais ne pouvant prendre que 2 valeurs : Vrai (1) et Faux (0) ou en Python **True** et **False** .

Ce type est très important en Python et aussi dans tous les autres langages de programmation.

Exemple :

```
print(10 > 5)
print(10 < 5)
```

Là encore la 1ère expression est vraie donc affichage de « True » et la 2ème est fausse donc affichage de « False » :



II. Exercices

Exercice 1

Quels sont les types des 2 données suivants ?
"Hello ", "007"

Exercice 2

Quels sont les types des littéraux suivants ?
"1.5", 2.0, 528, False

Exercice 3
Quelle est la valeur décimale du nombre binaire suivant ?
1011
Comment faire la conversion en python ?

III. Les opérations et Les opérateurs :

1- Opérateurs de base :

Comme tout langage de programmation qui se respecte, Python reconnaît les opérateurs arithmétiques les plus reconnus :

+ addition, - soustraction, * multiplication, / division, // division entière, % reste de la division euclidienne, ** exposant

Lorsqu'on regroupe des **données** et des **opérateurs** cela forme des **expressions**.

2- Opérateur d'exponentiation ** :

Le signe ** (double étoile) est un opérateur d'exponentiation (puissance). Son argument de gauche est la base, son argument de droite, l'exposant : 23 s'écrit 2 ** 3 en Python.

```
print(2 ** 3)
print(2 ** 3.)
print(2. ** 3)
print(2. ** 3.)
```

D'où : lorsque les deux arguments de l'opérateur « ** » sont des entiers, le résultat est également un entier ; lorsqu'au moins un des arguments est un float, le résultat est également un float.

3- Opérateur de multiplication * :

```
print(2 * 3)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
```

Son comportement est similaire à l'opérateur d'exponentiation « ** »

4- Opérateur de division / :

6 / 3 – 6 : dividende et 3 : diviseur

print(6 / 3)

print(6 / 3.)

print(6. / 3)

print(6. / 3.)

Nous constatons que le résultat de la division est toujours un float !

Justement, lorsqu'on souhaite que le résultat d'une division soit entier, on fait appel à :

5- Opérateur de division entière // :

print(6 // 4)

print(6. // 4)

print(-6 // 4)

print(6. // -4)

On pourrait croire que -6 // 4 = -1 (car -6 / 4 = -1.5), mais non, -6 // 4 = -2 !

Le résultat de la division des nombres entiers est toujours arrondi à la valeur entière la plus proche qui est inférieure au résultat réel (non arrondi).

C'est très important : l'arrondi se fait toujours sur **le nombre entier le plus petit**.

6- Opérateur : reste de la division (modulo) % :

6 % 4 = 2, car 6 / 4 = 1 et le reste de la division est justement 2 (6 = 4 * 1 + 2).

print(14 % 4)

print(12 % 4.5)

En effet, 14 / 4 = 3 et le reste est 2. 12 / 4.5 = 2 et le reste est 3 (2 * 4.5 = 9.0).

Remarque : Python ne permet pas n'importe quelle division par zéro.

7- Opérateur d'addition + :

print(-4 + 4)

print(-4. + 8)

8- Opérateur de soustraction - les opérateurs unaire et binaire :

L'opérateur « - » peut être utilisé de manière **binaire** (entre 2 opérandes) : 4 – 2 = 2, ou de manière **unaire** pour changer le signe d'un nombre : -2 (l'opposé de 2).

print(-4 - 4)

print(4. - 8)

print(-1.1)

print(+2)

9- Opérateur pour les chaines de caractères :

Concaténation : l'opérateur **« + »** permet de concaténer 2 chaines de caractère

Répétition : L'opérateur « * » permet de répéter la chaine de caractère un nombre de fois

Exemple :

```
>>> print("TDI"*2)
TDITDI
```

Ordre de priorité des opérateurs :

On considère l'exemple suivant où ne met aucune parenthèse. Est-ce que le résultat est 2 + (3*5) = 17 ou (2+3)*5 = 25. 33

print(2 + 3 * 5)

Les opérateurs et leurs liaisons :

A priorité égale, on effectue généralement les calculs de gauche à droite pour tous les opérateurs sauf l'exponentiation « ** ».

print(9 % 6 % 2)

En effet, de gauche à droite : 9 % 6 = 3 et 3 % 2 = 1

Python ne fait pas 6 % 2 = 0 et ensuite on sera embêté avec 9 % 0 !

Pour l'opérateur d'exponentiation, c'est différent :

print(2 ** 2 ** 3)

Python fait le traitement de droite à gauche, d'abord 2 ** 3 = 8, ensuite 2 ** 8 = 256

Et non pas de gauche à droite : 2 ** 2 = 4, ensuite 4 ** 3 = 64

Recapitulation :

Symbole	Opération	Type	Exemple
+	Addition	Entiers, réels, chaines de caractères	6 + 4 → 10 "TDI" + "2020" → "TDI2020"
-	Soustraction	entier, réels	6 – 4 → 2
*	Produit	entier, réels, chaines de caractères	6 * 4 → 24 1.2 * 2 → 2.4 3 * "TDI« → TDITDITDI
**	Puissance	Entiers, réels	12 ** 2 → 144
/	Division	Entiers, réels	6 / 4 = 1.5
//	Division entière	Entiers, réels	6 // 4 → 1
%	Modulo	Entiers, réels	6 % 4 → 2

IV. Priorités des opérations :

Priorité décroissante	Opérateur	Genre
1	+ , -	Unaire
2	**	
3	* , / , %	
4	+ , -	Binaire

+ et - unaires ont la priorité la plus élevée.

Ensuite **, suivi de * , / , et %, puis la priorité la plus faible : + et – binaire.

print(2 * 3 % 5)

En effet, * est prioritaire sur %, donc 2 * 3 = 6 % 5 = 1

Et non pas, 3 % 5 = 3 * 2 = 6 !

Opérateurs et parenthèses

Python permet bien évidement l'utilisation des parenthèses et les sous-expressions entre parenthèses sont prioritaires.

print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)

En effet, 25%13 = 12 + 100 = 112 / 26 (2*13) = 4.307… * 5 = 21.538…// 2 = 10.0

V. Exercices

Exercice 1 :

Essayez de prédire le résultat de chacune des instructions suivantes, puis vérifiez-le dans l'interpréteur Python :

- (1+2)**3
- "Da" * 4
- "Da" + 3
- ("Pa"+"La") * 2
- ("Da"*4) / 2
- 5 / 2
- 5 // 2
- 5 % 2

Exercice 2 :

Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((2 ** 4), (2 * 4.), (2 * 4))
```

Exercice 3 :

Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((-2 / 4), (2 / 4), (2 // 4), (-2 // 4))
```

Exercice 4 :

Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((2 % -4), (2 % 4), (2 ** 3 ** 2))
```

VI. Les variables

Rappel : Une variable est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom, alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.

Avec Python on est capable de stocker des données contenant des valeurs numériques et textuelles sur lesquelles on peut effectuer des opérations arithmétiques.

Tout de même , on les stocke dans des variables mais **En Python,** la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps.

on a pas besoin de préciser le type d'une variable car la variable en python prend automatiquement le type de la valeur qu'on va stocker dans cette variables

➔ En Python une variable doit avoir :
- Un nom ;
- Une valeur (le contenu du conteneur).

Exemple :

```
a = 2
print(a)
```

Ligne 1. Dans cet exemple, nous avons déclaré, puis initialisé la variable x avec la valeur 2. Notez bien qu'en réalité, il s'est passé plusieurs choses :

- Python a « deviné » que la variable était un entier. On dit que Python est un langage au typage dynamique.
- Python a alloué (réservé) l'espace en mémoire pour y accueillir un entier. Chaque type de variable prend plus ou moins d'espace en mémoire. Python a aussi fait en sorte qu'on puisse retrouver la variable sous le nom x.
- Enfin, Python a assigné la valeur 2 à la variable x.

Opérateur d'affectation :

Pour affecter ou "assigner" une valeur à une variable, nous allons utiliser un opérateur qu'on appelle opérateur d'affectation ou d'assignation et qui est

représenté par le signe =. Attention, le signe = ne signifie pas en informatique l'égalité d'un point de vue mathématique : c'est un opérateur d'affectation.

Le signe = ne sert pas à dire que la valeur est égale au nom de variable ou que la variable "vaut" cette valeur, il indique simplement qu'on affecte ou qu'on stocke une certaine valeur dans un conteneur.

Mais il existe d'autres types d'affectation qui nous permet d'affecter directement et automatiquement le résultat d'une opération

Opérateur	Exemple	Equivalent à	Description
=	x = 1	x = 1	Affecte 1 à la variable x
+=	x += 1	x = x + 1	Ajoute 1 à la dernière valeur connue de x et affecte la nouvelle valeur (l'ancienne + 1) à x
-=	x -= 1	x = x – 1	Enlève 1 à la dernière valeur connue de x et affecte la nouvelle valeur à x
*=	x *= 2	x = x * 2	Multiplie par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
/=	x /= 2	x = x / 2	Divise par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
%=	x %= 2	x = x % 2	Calcule le reste de la division entière de x par 2 et affecte ce reste à x
//=	x //= 2	x = x // 2	Calcule le résultat entier de la division de x par 2 et affecte ce résultat à x
**=	x **= 4	x = x ** 4	Elève x à la puissance 4 et affecte la nouvelle valeur dans x

Type des Variables :

Puisque Pyhton est un language de typage dynamique alors les types des variables sont de meme les types des données qu'on y stocke est alors les types des variables sont aussi 4 :

- Les entiers (*integer* ou *int*),
- Les nombres décimaux que nous appellerons *floats*
- Les chaînes de caractères (*string* ou *str*).
- Les booléens qui accepte deux valeurs (true ou False)

Exemple :

```
# declaration d'un entier
x = 3

#declaration d'un nombre decimale
y = 3.14

#declaration d'une chaine de caracteres
z1 = "TDI 2020"
z2 = 'TDI 2020'

#declaration d'un boolean
t1 = True
t2 = False
```

Les règles de nommage :

1- Le nom des variables en Python peut être constitué de lettres minuscules (a à z), de lettres majuscules (A à Z), de nombres (0 à 9) ou du caractère souligné (_). Vous ne pouvez pas utiliser d'espace dans un nom de variable.

2- Un nom de variable ne doit pas débuter par un chiffre.

3- De plus, il faut absolument éviter d'utiliser un mot « réservé » par Python comme nom de variable (par exemple : print, range, for, from, etc.).

4- Python est sensible à la casse :

   a. Ce qui signifie que les variables TesT, test ou TEST sont différentes.

Affichage des variables :

La fonction print() peut également afficher le contenu d'une variable quel que soit son type. Voici des exemples :

```
#Exemple 1
var = 3
print(var)

#Exemple 2
age = 32
name1 = 'zaid'
print(name1, "a",age,"ans")

#Exemple 3
note = 13.1
name2 = 'Riyad'
print("{0} a eu {1:.2f} en mathematique".format(name2,note))
```

Exercice d'Apprentissage :

Il était une fois à Appleland, Jean avait trois pommes, Marie cinq pommes et Adam six pommes. Ils étaient tous très heureux et ont vécu longtemps. Fin de l'histoire.

1- Créer les variables : john, mary, et adam ;
2- Attribuer des valeurs aux variables. Les valeurs doivent être égales au nombre de fruits possédés par Jean, Marie et Adam respectivement ;
3- Après avoir enregistré les nombres dans les variables, imprimez les variables sur une ligne et séparez-les par une virgule ;
4- Créez maintenant une nouvelle variable appelée totalPommes, égale à l'addition des trois anciennes variables.
5- Imprimez la valeur stockée dans totalApples sur la console ;
6- Expérimentez votre code : créez de nouvelles variables, attribuez-leur différentes valeurs et effectuez diverses opérations arithmétiques sur elles (par exemple, +, -, *, /, //, etc.).

VII. La fonction : Type() :

La fonction type retourne le type de données d'un objet quelconque.

Si vous ne vous souvenez plus du type d'une variable, utilisez la fonction **type()** qui vous le rappellera.

```
x = 3
y = 4.5
t = "TDI 2020"
u = False

print("Le type de x est : ", type(x))
print("Le type de y est : ", type(y))
print("Le type de t est : ", type(t))
print("Le type de u est : ", type(u))
```

Le type de x est : <class 'int'>
Le type de y est : <class 'float'>
Le type de t est : <class 'str'>
Le type de u est : <class 'bool'>

La conversion des types :

En programmation, on est souvent amené à convertir les types, c'est-à-dire passer d'un type numérique à une chaîne de caractères ou vice-versa. En Python, rien de plus simple avec les fonctions int(), float() et str().

Voici un exemple d'utilisation :

```
x = "3.14"
y = float(x)

print("La valeur de x est : ",x)
print("La valeur de y est : ",y)

print("Le type de x est : ",type(x))
print("Le type de y est : ",type(y))
```

La valeur de x est : 3.14
La valeur de y est : 3.14
Le type de x est : <class 'str'>
Le type de y est : <class 'float'>

Chapitre 4 : Fonction de Lecture : Input

I. La Fonction « Input » :

Dans le chapitre précédent on a vu que pour afficher des message ou des variables ou des résultats à l'utilisateur on a utilisé la fonction **print()** mais si on veut demander à l'utilisateur de saisir un texte ou une valeur ?

Alors pour cela on utilise la Fonction **Input()**

Cette fonction demande à l'utilisateur d'entrer une donnée qui va etre directement affectée à une variable , pour qu'on puisse récupérer ce que l'utilisateur a écrit. Mais il faut savoir que cette fonction retourne toujours une chaine de caractère ça veut dire qu'elle considère tout ce que l'utilisateur entre est une chaine de caractère même s'il entre une valeur numérique

Attention : par défaut, cette valeur est de type "string" ou "chaîne de caractère. si vous voulez un autre type de variable, il faudra la convertir.

Et pour le convertir on utilise les fonctions de conversion qu'on a vu précédemment selon le type voulu .

Exemples :

Exemple 1 : La fonction input sans argument

Code

```
entre = input()
print("Windows est un ", entre)
```

Console

```
Système d'exploitation
Windows est un Système d'exploitation
```

Le programme invite l'utilisateur à saisir certaines données à partir de la console. Ici input() est sans arguments (c'est la manière la plus simple d'utiliser la fonction).

Vous devez affecter le résultat à une variable ''mémoire''

Toutes les données saisies seront envoyées à votre programme via le résultat de la fonction

Exemple 2 : la fonction input avec argument

Code

```
entre = input("Windows, c'est quoi
au juste ?")
print("Windows est un", entre)
```

Console

```
Windows, c'est quoi au juste ?Système d'exploitation
Windows est un Système d'exploitation
```

Le message sera affiché sur la console avant d'entrer quoi que ce soit.

Le résultat de la fonction input() est une chaîne : Vous ne devez pas l'utiliser comme argument d'une opération arithmétique

Exemple 3 : Operations interdites

Code

```
nbr1 = input("Entrer a
nombre: ")
result = nbr1 + 2
print("le résultat est :",
result)
```

Console

```
Entrer a nombre: 5
Traceback (most recent call last):
  File "D:/Teams_Formation/Cours_py/venv/lab_LstInLst.py", line 39, in <module>
    result = nbr1 + 2
TypeError: can only concatenate str (not "int") to str
```

Questions : comment saisir les entiers et les nombres décimales a l'aide de la fonction input ?

Réponses : Python propose deux fonctions simples pour spécifier un type de données et résoudre ce problème : int() et float().

Exemple 4 : utilisation de la fonction int() et float()

Code

```
nbr1 = int(input("Entrer un nombre: "))
result = nbr1 + 2
print("le produit est :", result)
```

Console

```
Entrer a nombre: 5
le produit est : 7
```

Code

```
nbr1 = float(input("Entrer a nombre: "))
result = nbr1 * 2.0
print("le produit est :", result)
```

Console

```
Entrer a nombre: 2.5
le produit est : 5.0
```

II. Les Commentaires

Dans les langages de programmation souvent les développeurs ont tendance d'écrire quelque chose dans le code comme rappel ou commentaire mais ils ne

veulent pas que ces derniers se trompent avec le code donc on a inventé la notion des commentaires

En Python, nous insérons un commentaire sur une seule ligne avec le caractère **# (un signe dièse).**

Si nous voulons insérer un commentaire sur plusieurs lignes en Python, nous utilisons le symbole des guillemets doubles. Exemple :

```
"""
Ceci est un commentaire
en plusieurs lignes
qui sera ignoré lors de l'exécution
"""
```

III. Exercices :

Exercice 1 :

Ecrire un programme qui calcule et affiche a+b, a-b, a*b, a/b, a%b.

a et b sont saisi par l'utilisateur

Exercice 2 :

Ecrire un programme qui calcule la surface d'un cercle.

Exercice 3 :

Le travail consiste à écrire un code Python qui permet d'évaluer l'expression suivante :

$$\cfrac{1}{x+\cfrac{1}{x+\cfrac{1}{x+\cfrac{1}{x}}}}$$

Sachant que la variable x est une valeur réelle saisie par l'utilisateur et le résultat doit être attribué à une variable y

Exercice 4 :

Écrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement. (Prix total TTC = prix HT * Nombre article * (1 + taux TVA).

Exercice 5 : Ecrire un programme qui demande deux nombre A et B et qui :

- Calcule est affiche le carré de A et de B.
- A^B (A à la puissance B).
- La tangente de A en n'utilisant que les fonctions sin et cos.

Exercice 6 :

Un magasin dispose de cinq produits : Produit A : prix 5.00 DH - Produit B : prix 2.50 DH- Produit C : prix 3.00 DH -Produit D : prix 10.00 DH - Produit E : prix 7.00 DH

Un client achète : X unités du produit A. -Y unités du produit B. - Z unités du produit C. - T unités du produit D. - U unités du produit E.

On désire calculer et afficher :

- Le prix hors taxe (PHT) de cette vente.
- La taxe sur la valeur ajoutée (TVA)
- Le prix toutes taxes comprises (PTTC) de cette vente

On donne le taux de TVA : TTVA=0.20

Travail à faire :

Écrire un programme qui permet de calculer et afficher : Le prix hors taxe, La taxe sur la valeur ajoutée, Le prix toutes taxes comprises

Chapitre 5 : Structures Alternatives

I. Les Tests :

1- Opérateurs de comparaison :

Pour tester en python si une expression est fausse ou vraie il faut faire des comparaisons Et pour cela il faut avoir les opérateurs de comparaison :

- **L'opérateurs d'égalité :**

Pour comparer si deux valeurs ou deux variables sont égaux on utilise l'opérateur « == »

L'opérateur « == » (égal à) compare les valeurs de deux opérandes. S'ils sont égaux, le résultat de la comparaison est True. S'ils ne sont pas égaux, le résultat de la comparaison est False.

Il s'agit d'un opérateur binaire avec liaison à gauche. Il a besoin de deux arguments et vérifie s'ils sont égaux.

Exemple :

```
>>> a= 5
>>> b =6
>>> a == b
False
```

```
>>> a = 5
>>> b = 5
>>> a == b
True
```

Attention : ne pas confondre « = » avec « == »
« = » est un **opérateur d'affectation** , par exemple, a = b assigne avec la valeur de b;
« == »répond à la question *:ces valeurs sont-elles égales?* ; a == b **compare** a et b.

- **L'opérateurs non égalité : Différence :**

L'opérateur « **!=** » (différent de) compare également les valeurs de deux opérandes. Voici la différence : s'ils sont égaux, le résultat de la comparaison est False. S'ils ne sont pas égaux, le résultat de la comparaison est True.

Exemple :

```
>>> a = 6
>>> b = 5
>>> a != b
True
```

```
>>> a = 6
>>> b = 6
>>> a!= b
False
```

- **Les opérateurs relationnels :**

« > » strictement supérieur à

« < » strictement inférieur à

« >= » supérieur ou égale

« <= » inférieur ou égale

Table de priorité des opérateurs :

Nous devons maintenant mettre à jour notre **table de priorités** et y mettre tous les nouveaux opérateurs. Il se présente maintenant comme suit:

Priorité	Opérateur	
1	+, –	unaire
2	**	
3	*, /, //,%	
4	+, –	binaire
5	<, <=, >,>=	
6	==, !=	

2- Les opérateurs logiques :

Pour faire des tests complexes, Python aussi nous permet de faire des conditions composées en utilisant des opérateurs logiques pour soit la conjonction ou la disjonction des conditions .

Ces opérateurs sont :

- **And :**

Il s'agit d'un **opérateur binaire avec une priorité inférieure à celle exprimée par les opérateurs de comparaison**. Il nous permet de coder des conditions complexes sans utiliser de parenthèses comme celle-ci:

```
counter > 0  and  value == 100
```

Le résultat fourni par l'opérateur and peut être déterminé sur la base de la table de vérité.

Si nous considérons la conjonction de A and B, l'ensemble des valeurs possibles des arguments et des valeurs correspondantes de la conjonction se présente comme suit:

Argument A	Argument B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

- **Or :**

Un opérateur de disjonction est le mot or. C'est un opérateur binaire avec une priorité inférieure à and (tout comme + par rapport à *). Sa table de vérité est la suivante:

Argument A	Argument B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

- **Not :**

De plus, il existe un autre opérateur qui peut être appliqué pour construire des conditions. C'est un **opérateur unaire effectuant une négation logique**. Son fonctionnement est simple: il transforme la vérité en mensonge et le mensonge en vérité.

Cet opérateur est écrit comme le mot not, et **sa priorité est très élevée : la même chose que les unaires** + **et** -. Sa table de vérité est simple

Remarque : Loi de Morgan :

```
not (p and q) == (not p) or (not q)
not (p or q) == (not p) and (not q)
```

II. Exercice :

Quelle est la sortie de l'extrait de code suivant?

x = 1

y = 0

z = ((x == y) and (x == y)) or not(x == y)

print(not(z))

III. Les conditions et les Exécutions conditionnelles :

L'instruction IF :

La première forme d'une déclaration conditionnelle, que vous pouvez voir ci-dessous, est écrite de manière très informelle mais figurative:

```
if condition :
    #Traitement à faire si la condition est vrai
```

Cette déclaration conditionnelle comprend uniquement les éléments suivants, strictement nécessaires:

- le mot - clé **if**
- un ou plusieurs espaces blancs

- une expression (une question ou une réponse) dont la valeur sera interprétée uniquement en termes de True (lorsque sa valeur est non nulle) et False (lorsqu'elle est égale à zéro)
- un **colon** suivi d'un saut de ligne
- une instruction en **retrait** ou un ensemble d'instructions (au moins une instruction est absolument requise); l' **indentation** peut être obtenue de deux manières - en insérant un nombre particulier d'espaces (la recommandation est d'utiliser **quatre espaces d'indentation**), ou en utilisant le caractère de *tabulation* ; note: s'il y a plus d'une instruction dans la partie en retrait, l'indentation doit être la même sur toutes les lignes; même s'il peut avoir la même apparence si vous utilisez des tabulations mélangées à des espaces, il est important que toutes les indentations soient **identiques** - Python 3 **ne permet pas de mélanger les espaces et les tabulations** pour l'indentation.

Exemple :

```
a = 100
if a < 100 :
    print(a,"est inférieur à 100")
print("BYE")
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
BYE
```

L'instruction IF .. ELSE :

Pour exprimer le plan alternatives cad qu'est ce qu'on va faire si la condition n'est pas vrai python utilise l'instruction **if..else** (sinon on algorithme)

```
if condition :
        #Traitement à exécuter si la condition est vrai
else:
        #Traitement à exécuter si la condition est fausse
```

Exemple :

```
a = 100
if a < 100 :
    print(a,"est inférieur à 100")
else:
    print(a," est supérieur ou égale à 100")

print("BYE")
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
100  est supérieur ou égale à 100
BYE
```

Les tests imbriqués :

On peut insérer d'autres tests au sein d' un test :

If condition1 **:**
    If condition2 **:**
        #traitement à éxecuter
    Else :
        #traitement inverse
Else :
    #traitement à éxecuter si condition1 est fausse

Exemple :

```
a = 100
if a != 100 :
    if a<100 :
        print(a,"est inférieur à 100")
    else:
        print(a,"est supérieur à 100")
else:
    print(a," est égale à 100")

print("BYE")
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
100  est égale à 100
BYE
```

L'instruction if .. elif .. else :

Python propose une autre instruction pour gérer les exécutions conditionnelle c'est l'instruction **elif** qui est équivalent à **else if**

If condition1 **:**

    #traitement à exécuter

Elif condition 2 **:**

    #traitement à exécuter

Elif condition 3 **:**

    #traitement à exécuter

Else :

    #traitement inverse

Exemple :

```
a = 90
if a == 100 :
    print(a," est égale à 100")
elif a<100 :
    print(a,"est inférieur à 100")
else:
    print(a,"est supérieur à 100")
print("BYE")
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
90 est inférieur à 100
BYE
```

IV. Exercices :

Exercice 1 :

Écrire un algorithme qui permet la résolution d'une équation du second degré (une équation sous la forme $ax^2+bx+c=0$)

Exercice 2 :

Calculer le lendemain d'une journée donnée (jour, mois, année)" On ne tiendra pas compte ici des années bissextiles, le mois de février aura toujours 28 jours.

Exercice 3 :

Calculer la durée d'un trajet connaissant l'heure de départ et d'arrivée". On se contente des heures et des minutes, la durée totale ne dépassera jamais 24 heures.

Exercice 4 :

La direction d'un supermarché a décidé d'accorder des réductions à ses clients selon le montant d'achat.

La réduction est calculée selon les règles suivantes:

- 20% pour un montant d'achat de plus de 5000 dhs
- 15% pour un montant d'achat entre 3000 dhs<montant d'achat <= 5000 dhs
- 10% pour un montant d'achat entre 1000 dhs <montant d'achat <= 3000 dhs
- Aucune réduction pour un montant d'achat inférieur à 1000 dhs.

Ecrire un programme qui permet de calculer et d'afficher la réduction et montant à payer.

Exercice 5 :

Le service des prêts d'une bibliothèque à adapter le règlement suivant :

- Tous les lecteurs de la catégorie A, peuvent emprunter des livres pour une durée de 20 jours.
- Un lecteur de la catégorie B, peut conserver des livres pour une durée de 30 jours.
- Le lecteur de la catégorie C, peut conserver les livres empruntés pendant 45 jours.
- Aucun lecteur ne pourra avoir en sa possession plus de 5 ouvrages et cela quelque soit sa catégorie.

Ecrire un programme où on affiche : la durée d'emprunt et est-ce qu'il a le droit d'emprunter

Chapitre 6 : Structures Répétitives

I. Les structures Répétitives :

Pour répéter un traitement un nombre de fois que ce soit connu ou inconnu les languages de programmation on recours à des structures qu'on appelle des boucles . Python n'est pas une exception , et alors python aussi a ces propres boucles

1- Boucle While (tantque) :

Syntaxe :

```
while condition :

    instruction_one
    instruction_two
    instruction_three
    …
    instruction_n
```

Il est maintenant important de se rappeler que:

- si vous voulez exécuter **plus d'une instruction à l'intérieur d'une** while , vous devez (comme avec if) mettre en **retrait** toutes les instructions de la même manière
- une instruction ou un ensemble d'instructions exécutées à l'intérieur de la boucle while est appelé **corps de** la **boucle**
- si la condition est False (égale à zéro) dès lors qu'elle est testée pour la première fois, le corps n'est pas exécuté une seule fois (notez l'analogie de ne rien faire s'il n'y a rien à faire)
- le corps devrait être en mesure de modifier la valeur de la condition, car si la condition est True au début, le corps peut fonctionner en continu à

l'infini - notez que faire une chose diminue généralement le nombre de choses à faire).

Exemple 1 :

```
x=int(input("Entrez un nombre"))
while x!=0:
    x= int(input("Entrez un nombre"))
print("BYE")
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
Entrez un nombre 5
Entrez un nombre 3
Entrez un nombre 8
Entrez un nombre 7
Entrez un nombre 3
Entrez un nombre0
BYE
```

Exemple 2 :

```
#Programme qui calcule combien de nombre paires entrés par l'utilisateur
#Et combien de nombre impaires
nbre_paires = 0
nbre_impaires = 0
nombre=int(input("Entrez un nombre ou tapez 0 pour sortir :"))
while nombre!=0:
    if nombre % 2 == 0 :
        nbre_paires +=1
    else:
        nbre_impaires+=1
    nombre= int(input("Entrez un nombre ou tapez 0 pour sortir :"))
print("le nombre des nombres paires est :",nbre_paires)
print("Le nombre des nombres impaires est :",nbre_impaires)
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
Entrez un nombre ou tapez 0 pour sortir :5
Entrez un nombre ou tapez 0 pour sortir :2
Entrez un nombre ou tapez 0 pour sortir :16
Entrez un nombre ou tapez 0 pour sortir :11
Entrez un nombre ou tapez 0 pour sortir :18
Entrez un nombre ou tapez 0 pour sortir :0
le nombre des nombres paires est : 3
Le nombre des nombres impaires est : 2
```

Certaines expressions peuvent être simplifiées sans changer le comportement du programme.

Essayez de vous rappeler comment Python interprète la vérité d'une condition et notez que ces deux formes sont équivalentes :

while number != 0: et **while number :**

La condition qui vérifie si un nombre est impair peut également être codée sous ces formes équivalentes :

if number % 2 == 1: et **if number % 2:**

➢ La boucle infinie :

Une boucle infinie, également appelée **boucle sans fin** , est une séquence d'instructions dans un programme qui se répète indéfiniment (boucle sans fin.)

Voici un exemple de boucle qui n'est pas en mesure de terminer son exécution :

Exemple :

```
while True:
    print("Je suis coincé dans la boucle infinie")

print("BYE")
```

➢ Utilisation d'un compteur

Regardez l'extrait ci-dessous:

```
counter = 5
while counter != 0:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

Ce code est destiné à imprimer la chaîne "**Inside the loop**." et la valeur stockée dans la variable counter pendant une boucle donnée exactement cinq fois. Une fois que la condition n'est pas remplie (la variable counter atteint 0), la boucle est quittée et le message "**Outside the loop.**"ainsi que la valeur stockée dans countersont imprimés.

2- La Boucle For (Pour) :

Un autre type de boucle disponible en Python vient de l'observation qu'il est parfois plus important de **compter les "tours" de la boucle** que de vérifier les conditions.

Imaginez que le corps d'une boucle doit être exécuté exactement cent fois. Si vous souhaitez utiliser la boucle while pour le faire, cela peut ressembler à ceci:

```
i = 0
while i < 100:
# do_something()
i += 1
```

Ce serait bien si quelqu'un pouvait faire ce comptage ennuyeux pour vous. Est-ce possible?

Bien sûr que oui - il y a une boucle spéciale pour ce genre de tâches, et elle est nommée **for**.

En fait, la boucle for est conçue pour effectuer des tâches plus compliquées - **elle peut «parcourir» de grandes collections de données élément par élément** . Nous allons vous montrer comment faire cela bientôt, mais pour le moment, nous allons présenter une variante plus simple de son application.

Jetez un oeil à l'extrait:

```
for i in range(100):
# do_something()
```

Il y a de nouveaux éléments. Laissez-nous vous en parler :

- le mot-clé *for* ouvre la boucle for note - il n'y a aucune condition après cela vous n'avez pas à penser aux conditions, car elles sont vérifiées en interne, sans aucune intervention;
- toute variable après le mot clé *for* est la **variable** de **contrôle** de la boucle (compteur) il compte les tours de boucle et le fait automatiquement

• le mot clé *in* introduit un élément de syntaxe décrivant la plage de valeurs possibles affectées à la variable de contrôle

• la fonction range()(c'est une fonction très spéciale) est chargée de générer toutes les valeurs souhaitées de la variable de contrôle; dans notre exemple, la fonction créera (on peut même dire qu'elle **alimentera** la boucle avec) les valeurs suivantes de l'ensemble suivant: 0, 1, 2 .. 97, 98, 99; note: dans ce cas, la fonction range() démarre son travail à partir de 0 et le termine une étape (un nombre entier) avant la valeur de son argument;

Nos prochains exemples seront un peu plus modestes quant au nombre de répétitions de boucle.

Exemple :

```
for i in range(10):
print("The value of i is currently", i)
```

Exécutez le code pour vérifier si vous aviez raison.

Remarque:

• la boucle a été exécutée dix fois (c'est l'argument de la fonction range())

• la dernière valeur de la variable de contrôle est 9(pas 10, car **elle commence à partir** 0 , pas à partir de 1)

- **La fonction Range :**

- Range () est l'appel de cette fonction qui peut être équipée de deux arguments, pas d'un seul:

```
for i in range(2, 8):
print("The value of i is currently", i)
```

Dans ce cas, le premier argument détermine la (première) valeur initiale de la variable de contrôle.

Le dernier argument montre la première valeur à laquelle la variable de contrôle ne sera pas affectée.

Remarque: la range() fonction **accepte uniquement des entiers comme arguments** et génère des séquences d'entiers.

Pouvez-vous deviner la sortie du programme ? Exécutez-le pour vérifier si vous étiez en ce moment aussi.

La première valeur affichée est 2(tirée du range ()premier argument de.)

Le dernier est 7(bien que le range() deuxième argument de soit 8).

- La fonction range() peut également accepter **trois arguments**

```
for i in range(2,8,3):
    print("hello")
print("BYE")
```

Le troisième argument est le **pas** - c'est une valeur ajoutée pour contrôler la variable à chaque tour de boucle (comme vous vous en doutez, la **valeur par défaut du pas est 1**).

Résultat :

```
==== RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\d$.py ====
hello
hello
BYE
```

Remarque: si l'ensemble généré par la range()fonction est vide, la boucle n'exécutera pas du tout son corps.

Tout comme ici - il n'y aura pas de sortie:

```
for i in range(1, 1):
print("The value of i is currently", i)
```

Remarque: l'ensemble généré par le range()doit être trié **par ordre croissant** . Il n'y a aucun moyen de forcer le range()à créer un ensemble sous une forme différente. Cela signifie que le range()deuxième argument de doit être supérieur au premier.

Ainsi, il n'y aura pas de sortie ici non plus:

```
for i in range(2, 1):
print("The value of i is currently", i)
```

3- Les déclarations Break et Continue

Jusqu'à présent, nous avons traité le corps de la boucle comme une séquence d'instructions indivisible et inséparable qui sont exécutées complètement à chaque tour de la boucle. Cependant, en tant que développeur, vous pourriez être confronté aux choix suivants :

• il semble qu'il ne soit pas nécessaire de continuer la boucle dans son ensemble; vous devez vous abstenir de poursuivre l'exécution du corps de la boucle et aller plus loin;

• il semble que vous devez commencer le tour suivant de la boucle sans terminer l'exécution du tour en cours.

Python fournit deux instructions spéciales pour l'implémentation de ces deux tâches. Ces deux instructions sont les suivantes:

• **break**- quitte la boucle immédiatement et met fin inconditionnellement au fonctionnement de la boucle; le programme commence à exécuter l'instruction la plus proche après le corps de la boucle;

• **continue**- se comporte comme si le programme avait soudainement atteint la fin du corps; le tour suivant est commencé et l'expression de la condition est testée immédiatement.

Ces deux mots sont des **mots clés**.

Exemple :

```
# break - example
print("The break instruction:")
for i in range(1, 6):
    if i == 3:
        break
    print("Inside the loop.", i)
print("Outside the loop.")
# continue - example
print("\nThe continue instruction:")
for i in range(1, 6):
    if i == 3:
        continue
    print("Inside the loop.", i)
print("Outside the loop.")
```

Résultat :

```
==== RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\d$.py ==
The break instruction:
Inside the loop. 1
Inside the loop. 2
Outside the loop.

The continue instruction:
Inside the loop. 1
Inside the loop. 2
Inside the loop. 4
Inside the loop. 5
Outside the loop.
```

4- L'instruction else :

Les boucles **while** et **for** en python seulement ont aussi une instruction **else** comme if qui s'exécute **toujours une fois, que la boucle soit entrée dans son corps ou non**.

Exemple avec for :

```
for i in range(1,1):
    print(i)

else:
    print("ecrire else:")
```

Exemple avec while :

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

II. Exercices :

Exercice 1

Créez une for boucle qui compte de 0 à 10 et imprime des nombres impairs à l'écran.

Exercice 2

Créez une while boucle qui compte de 0 à 10 et imprime des nombres impairs à l'écran.

Exercice 3

Créez un programme avec une for boucle et une break instruction. Le programme doit parcourir les caractères d'une adresse e-mail, quitter la boucle lorsqu'il atteint le @symbole et imprimer la partie précédente @sur une seule ligne.

Exercice 4

Créez un programme avec une for boucle et une continue instruction. Le programme doit parcourir une chaîne de chiffres, remplacer chacun 0par x et imprimer la chaîne modifiée à l'écran.

Chapitre 7 : Les Fonctions et les méthodes

I. Les Fonctions :

En programmation, les fonctions sont très utiles pour réaliser plusieurs fois la même opération au sein d'un programme. Elles rendent également le code plus lisible et plus clair en le fractionnant en blocs logiques.

Il nous est arrivé de travailler avec plusieurs fonctions sans qu'on nous rendons compte exemple : print() , input() , int() , type() , str() , range() , … donc les fonctions sont des pseudo-programmes qu'on définit à l'avance et qui font un traitement spécifique qu'on va utiliser après dans notre programme principale.

En python comme dans d'autres langages de programmation il existe plusieurs fonctions . Donc **d'où vient ces fonctions ?**

En général, les fonctions proviennent d'au moins trois endroits :

• à partir de Python lui-même - de nombreuses fonctions (comme print ()) font **partie intégrante de Python** et sont toujours disponibles sans effort supplémentaire de la part du programmeur ; nous appelons ces fonctions des **fonctions intégrées** ;

• à partir des **modules préinstallés** de Python - de nombreuses fonctions, très utiles, mais utilisées beaucoup moins souvent que celles intégrées, sont disponibles dans un certain nombre de modules installés avec Python ; l'utilisation de ces fonctions nécessite quelques étapes supplémentaires de la part du programmeur afin de les rendre entièrement accessibles (nous vous en parlerons dans un moment);

• **directement à partir de votre code** - vous pouvez écrire vos propres fonctions, les placer dans votre code et les utiliser librement ;

• il existe une autre possibilité, mais elle est liée aux classes, nous allons donc l'omettre pour l'instant.

1- Définition d'une fonction :

Pour définir une fonction, Python utilise le mot-clé **def :**

```
def message():
    print("Hello world")
```

Si on souhaite qu'elle renvoie ou retourne un résultat on ajoute le mot clé **return** :

```
def calcul_somme(x,y):
    return x+y
```

Notez que la syntaxe de **def** utilise les deux points comme les boucles for et while ainsi que les tests if, un bloc d'instructions est donc attendu. De même que pour les boucles et les tests, l'indentation de ce bloc d'instructions (qu'on appelle le corps de la fonction) est obligatoire

2- L'invocation de la Fonction (Appel) :

La fonction se déclare généralement avant le programme principale et pour l'appeler il suffit d'invoquer son nom au sein du programme principale

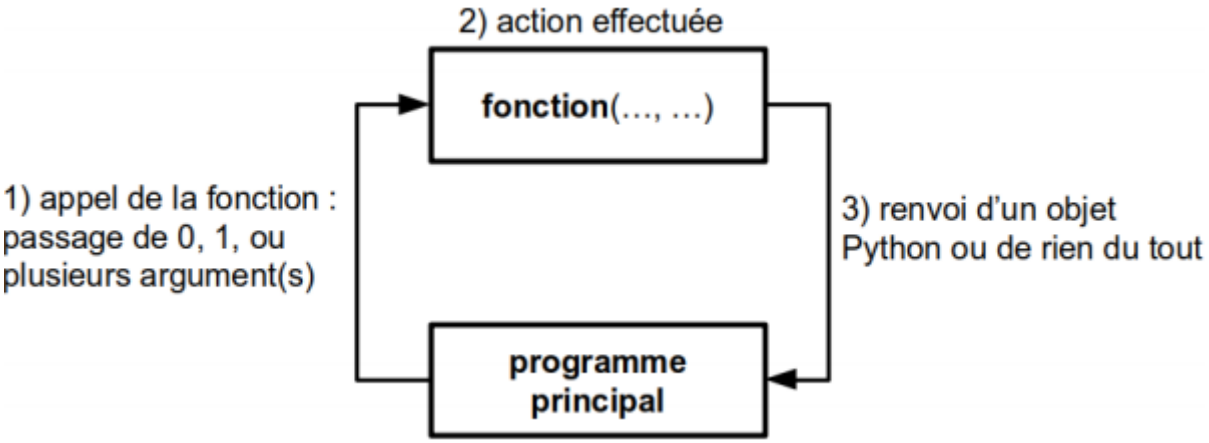
Exemple :

```
def message():
    print("Hello world")

def calcul_somme(x,y):
    return x+y

print("programme principale :")
message()
a=int(input("Entrez a :"))
b=int(input("Entrez b :"))
print("La somme est :",calcul_somme(a,b))
print("Bye")
```

Résultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\serie_srepeti
ttives.py
programme principale :
Hello world
Entrez a : 5
Entrez b : 3
La somme est : 8
Bye
```

Comment ça fonctionne ?



Dans l'exemple précédent, nous avons passé un argument à la fonction calcul_somme() qui nous a renvoyé (ou retourné) une valeur que nous avons immédiatement affichée à l'écran avec l'instruction print(). Que veut dire valeur renvoyée ? Et bien cela signifie que cette dernière est récupérable dans une variable

Dans ce cas la fonction, message() se contente d'afficher la chaîne de caractères "Hello world " à l'écran. Elle ne prend aucun argument et ne renvoie rien. Par conséquent, cela n'a pas de sens de vouloir récupérer dans une variable le résultat renvoyé par une telle fonction.

3- Passage d'arguments :

Le nombre d'arguments que l'on peut passer à une fonction est variable. Nous avons vu ci-dessus des fonctions auxquelles on passait 0 ou 1 argument. Dans les chapitres précédents, vous avez rencontré des fonctions internes à Python qui

prenaient au moins 2 arguments. Souvenez-vous par exemple de range(1, 10) ou encore range(1, 10, 2). Le nombre d'argument est donc laissé libre à l'initiative du programmeur qui développe une nouvelle fonction. Une particularité des fonctions en Python est que vous n'êtes pas obligé de préciser le type des arguments que vous lui passez, dès lors que les opérations que vous effectuez avec ces arguments sont valides. Python est en effet connu comme étant un langage au « typage dynamique », c'est-à-dire qu'il reconnaît pour vous le type des variables au moment de l'exécution.

Exemple :

```
>>> def fois(x,y):
        return x*y

>>> fois(2,3)
6
>>>
>>> fois(3.1456,5.892)
18.5338752
>>>
>>> fois('to',5)
'tototototo'
>>>
>>> fois([1,3],2)
[1, 3, 1, 3]
```

L'opérateur * reconnaît plusieurs types (entiers, floats, chaînes de caractères, listes). Notre fonction fois() est donc capable d'effectuer des tâches différentes.

4- Renvoi des resultats :

Un énorme avantage en Python est que les fonctions sont capables de renvoyer plusieurs objets à la fois, comme dans cette fraction de code :

```
>>> def carre_cube(x):
        return x**2,x**3

>>> carre_cube(2)
(4, 8)
```

En réalité Python ne renvoie qu'un seul objet, mais celui-ci peut être séquentiel, c'est-à-dire contenir lui-même d'autres objets. Dans notre exemple Python renvoie un objet de type tuple, type que nous verrons plus tard.

Notre fonction pourrait tout autant renvoyer une liste :

```
>>> def carre_cube(x):
        return [x**2,x**3]

>>> carre_cube(5)
[25, 125]
```

Remarque 1 :

si une fonction n'est pas destinée à produire un résultat, l' **utilisation de l'** return **instruction n'est pas obligatoire** - elle sera exécutée implicitement à la fin de la fonction.

Quoi qu'il en soit, vous pouvez l'utiliser pour **terminer les activités d'une fonction à la demande** , avant que le contrôle n'atteigne la dernière ligne de la fonction.

Remarque 2 :

On peut utiliser le mot clé return pour interrompre l'exécution d'une fonction dans ce cas il suffit d'écrire return ou return 0

Exemple :

```
def happy_new_year(corona):
    print("trois..")
    print("deux..")
    print("Un..")
    if not corona:
        return
    print("Happy New Year")

reponse=input("Est ce qu'il ya toujours corona ?")
if reponse =="OUI":
    corona=False
else:
    corona=True
happy_new_year(corona)
```

5- Arguments positionnels et arguments par mot-clé :

- **Arguments positionnels :**

Jusqu'à maintenant, nous avons systématiquement passé le nombre d'arguments que la fonction attendait. Que se passe-t-il si une fonction attend deux arguments et que nous ne lui en passons qu'un seul ?

```
>>> def fois(x,y):
        return x*y

>>> fois(2,3)
6
>>> fois(5)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    fois(5)
TypeError: fois() missing 1 required positional argument: 'y'
```

On constate que passer un seul argument à une fonction qui en attend deux conduit à une erreur.

Définition : Lorsqu'on définit une fonction def fct(x, y): les arguments x et y sont appelés arguments positionnels (en anglais positional arguments). Il est strictement obligatoire de les préciser lors de l'appel de la fonction. De plus, il est nécessaire de respecter le même ordre lors de l'appel que dans la définition de la fonction. Dans l'exemple ci-dessus, 2 correspondra à x et 3 correspondra à y. Finalement, tout dépendra de leur position, d'où leur qualification de positionnel.

- **Argument avec mot-clé :**

Python propose une autre convention pour le passage d'arguments, où la signification de l'argument est dictée par son nom, et non par sa position - c'est ce qu'on appelle le passage d'argument par mot clé.

Exemple :

```
def Introduction(firstname , lastname):
    print("Bonjour Je suis :",firstname,lastname)


print("Programme Principale")
Introduction("joairia","lafhal")
Introduction(firstname="joairia",lastname="lafhal")
Introduction(lastname="lafhal",firstname="joairia")
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Loc
ttives.py
Programme Principale
Bonjour Je suis : joairia lafhal
Bonjour Je suis : joairia lafhal
Bonjour Je suis : joairia lafhal
```

Le concept est clair - les valeurs transmises aux paramètres sont précédées du nom des paramètres cibles, suivi du signe =.

La position n'a pas d'importance ici - la valeur de chaque argument connaît sa destination sur la base du nom utilisé.

Remarque : je dois respecter l'écriture du mot-clé comme je l'ai écrit dans la définition du la fonction je dois l'écrire dans l'appel

- **Mélanger les arguments positionnels et avec mot clé :**

Vous pouvez mélanger les deux modes si vous le souhaitez - il n'y a qu'une seule règle incassable: **vous devez mettre les arguments positionnels avant les arguments mot-clé**.

Pour vous montrer comment cela fonctionne, nous utiliserons la fonction simple à trois paramètres suivants :

```
def adding(a,b,c):
    print(a,"+",b,"+",c,"=",a+b+c)
```

Son but est d'évaluer et de présenter la somme de tous ses arguments.

La fonction, lorsqu'elle est invoquée de la manière suivante :

adding(1, 2, 3)

affichera :

1 + 2 + 3 = 6

C'était - comme vous pouvez le soupçonner - un pur exemple de **passage d'arguments positionnels**.

Bien sûr, vous pouvez remplacer une telle invocation par une variante purement mot-clé, comme celle-ci :

adding(c = 1, a = 2, b = 3)

Notre programme affichera une ligne comme celle-ci :

2 + 3 + 1 = 6

Notez l'ordre des valeurs.

Essayons maintenant de mélanger les deux styles.

Regardez l'invocation de fonction ci-dessous :

adding(3, c = 1, b = 2) Analysons-le:

• l'argument (3) pour le paramètre a est passé en utilisant la manière positionnelle ;

• les arguments pour c et b sont spécifiés comme mots-clés.

Voici ce que vous verrez dans la console :

3 + 2 + 1 = 6

Soyez prudent et méfiez-vous des erreurs. Si vous essayez de passer plusieurs valeurs à un seul argument, vous n'obtiendrez qu'une erreur d'exécution.

Regardez l'invocation ci-dessous - il semble que nous ayons essayé de définir a deux fois :

```
>>> adding(3,a=2,b=1)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    adding(3,a=2,b=1)
TypeError: adding() got multiple values for argument 'a'
```

6- Les fonctions paramétrées (valeur par défaut) :

Il arrive parfois que les valeurs d'un paramètre particulier soient utilisées plus souvent que d'autres. Ces arguments peuvent avoir leurs **valeurs par défaut (prédéfinies)** prises en considération lorsque leurs arguments correspondants ont été omis.

Exemple :

```
def Introduction(lastname, firstname ="Mohamed"):
    print("Bonjour Je suis :",firstname,lastname)


print("Programme Principale")
Introduction("Lafhal")
Introduction("lafhal","joairia")
Introduction(firstname="joairia",lastname="lafhal")
Introduction(lastname="lafhal")
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppDat
ttives.py
Programme Principale
Bonjour Je suis : Mohamed Lafhal
Bonjour Je suis : joairia lafhal
Bonjour Je suis : joairia lafhal
Bonjour Je suis : Mohamed lafhal
```

On remarque que lorsqu'on appelé la fonction avec un seul paramètre ça n'a pas créé des problèmes car python a déjà une valeur par défaut pour le 2ème argument .

7- Les variables locales et globales :

Lorsqu'on manipule des fonctions, il est essentiel de bien comprendre comment se comportent les variables. Une variable est dite locale lorsqu'elle est créée dans une fonction. Elle n'existera et ne sera visible que lors de l'exécution de ladite fonction. Une variable est dite globale lorsqu'elle est créée dans le programme principal. Elle sera visible partout dans le programme.

Exemple :

```
def carre(x):
    return x**2

print("Programme principale")
z= int(input("Entrez un nombre :" ))
print(x)
y=carre(x)
print(y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\P
ttives.py
Programme principale
Entrez un nombre :5
Traceback (most recent call last):
  File "C:\Users\TOSHIBA\AppData\Local\Prog
ves.py", line 6, in <module>
    print(x)
NameError: name 'x' is not defined
```

Python ne connait pas la variable **x** car elle est créé dans la fonction et manipulée dans la fonction est détruite une fois le traitement de la fonction est terminé donc les variables locales n'ont aucun rôle en dehors la fonction.

Par contre les variables **z** et **y** sont des variables global elles sont connu partout dans le programme principale et même dans une fonction

Exemple :

```
def carre(x):
    print(a)
    return x**2

a=5
print("Programme principale")
z= int(input("Entrez un nombre :" ))
y=carre(z)
print(y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Pr
ttives.py
Programme principale
Entrez un nombre :6
5
36
```

Remarque : Python reconnait les variables globales dans la fonction mais on ne peut pas modifier la valeur de la variable dans le programme principale

Exemple :

```
def carre(x):

    print("Incrementation de a dans la fonction :")
    a+=1
    print(a)
    return x**2

a=5
print("Programme principale")
z= int(input("Entrez un nombre :" ))
y=carre(z)
print("a=",a)
print("y=",y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\
ttives.py
Programme principale
Entrez un nombre :6
Incrementation de a dans la fonction :
Traceback (most recent call last):
  File "C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\ser
ves.py", line 11, in <module>
    y=carre(z)
  File "C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\ser
ves.py", line 4, in carre
    a+=1
UnboundLocalError: local variable 'a' referenced before assignment
```

- **Le mot clé « global » :**

Il existe une méthode Python spéciale qui peut **étendre la portée d'une variable d'une manière qui inclut les corps des fonctions** (même si vous voulez non seulement lire les valeurs, mais aussi les modifier).

Un tel effet est provoqué par un mot-clé nommé global :

global name

global name1, name2, ...

L'utilisation de ce mot-clé dans une fonction avec le nom (ou les noms séparés par des virgules) d'une ou de plusieurs variables, force Python à s'abstenir de créer une nouvelle variable à l'intérieur de la fonction - celle accessible de l'extérieur sera utilisée à la place.

En d'autres termes, ce nom devient global (il a une **portée globale** et peu importe qu'il fasse l'objet d'une lecture ou d'une affectation).

Exemple :

```
def carre(x):
    global a
    print("Incrementation de a dans la fonction :")
    a+=1
    print(a)
    return x**2

a=5
print ("a avant fonction : ",a)
z= int(input("Entrez un nombre :" ))
y=carre(z)
print("a apres la fonction ",a)
print("y=",y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Prog
ttives.py
a avant fonction :  5
Entrez un nombre :6
Incrementation de a dans la fonction :
6
a apres la fonction  6
y= 36
```

8- La valeur None :

Il n'y a que deux types de circonstances qui None peuvent être utilisées en toute sécurité :

• lorsque vous l'**affectez à une variable** (ou le retournez comme résultat d'une **fonction**)

• lorsque vous le **comparez à une variable** pour diagnostiquer son état interne.

Comme ici:

```
value = None
if value is None:
print("Sorry, you don't carry any value")
```

N'oubliez pas ceci : si une fonction ne retourne pas une certaine valeur en utilisant une return clause d'expression, on suppose qu'elle **renvoie implicitement** None. (On peut l'utiliser dans l'exercice 6 pour tester si a et b sont saisis)

9- Les Fonctions récursives :

En mathématiques, une suite (Un) n∈N est récurrente lorsque le terme un+1 est une fonction du terme un. En informatique, une fonction f est récursive lorsque la définition de f utilise des valeurs de f. Chaque fonction récursive est construite sur une relation de récurrence.

Exemple :

Le factoriel : La fonction factorielle est une fonction récursive car elle appelle elle-même dans le traitement

Si on souhaite calculer le factoriel de 5 c'est

5 != 5*4*3*2*1

Et 4*3*2*1 c'est le factoriel de 4

Donc 5 !=5*4 !

Et donc 5 !=5*4*3 ! est ça continue comme ça donc en peut écrire

Exemple :

```
def factoriel(x):
    if x==0:
        return 1
    return x*factoriel(x-1)

print("Programme principale :" )
a=int(input("Entrez un nombre :"))
print("le resultat est :",factoriel(a))
```

Résultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Loca
ttives.py
Programme principale :
Entrez un nombre :5
le resultat est : 120
```

10- Passage par adresse et passage par valeur :

En python c'est toujours le passage des paramètres est par valeur pour les variables ça se fait automatiquement donc on n'aura pas de problèmes en python par contre il faut faire attention dans les autres langages de programmation

Exemple :

```
def f(x):
    x+=1
    print("dans la fonction:",x)

print("Programme principale :" )
a=2
print("a avant la fonction:",a)
f(a)
print("a apres la fonction:",a)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Lo
ttives.py
Programme principale :
a avant la fonction: 2
dans la fonction: 3
a apres la fonction: 2
```

On remarque que la modification de la variables ne se propage pas en dehors de la fonction est donc le passage des variables en parametre se fait automatiquement par valeur

11- Exercices

Exercice 1 :

créez une fonction gen_pyramide() à laquelle vous passez un nombre entier N et qui renvoie une pyramide de N lignes sous forme de chaîne de caractères. Le programme principal demandera à l'utilisateur le nombre de lignes souhaitées (utilisez pour cela la fonction input()) et affichera la pyramide à l'écran.

Exercice 2 :

Ecrire une fonction qui accepte comme paramètre le jour, le mois et l'année qui détermine si la date est correcte.

Exercice 3 :

Ecrire une fonction **chiffrePorteBonheur(nb)** qui permet de déterminer si un nombre entier **nb** est chiffre porte Bonheur ou non.

Un nombre **chiffre porte Bonheur** est un nombre entier qui, lorsqu'on ajoute les **carrés** de **chacun** de ses chiffres, puis les carrés des chiffres de ce **résultat** et ainsi de suite jusqu'à l'obtention d'un nombre à un seul chiffre égal à 1 (un).

Le calcule s'arrête lorsque le chiffre devient inférieur à 10

le résultat doit être comme suit :

```
Nombre de départ: 913
9^2=81
1^2=1
3^2=9
Nouveau: 81+1+9=91
9^2=81
1^2=1
Nouveau: 81+1=82
8^2=64
2^2=4
Nouveau: 64+4=68
6^2=36
8^2=64
Nouveau: 36+64=100
1^2=1
0^2=0
0^2=0
Nouveau: 1+0+0=1
Le chiffre: 913 est un porte boneur
```

Exercice 4 :

Définir une fonction qui renvoie les nombres parfaits qui se trouve entre deux nombres entiers a et b. tel que a < b.

Exercice 5 :

Définir une fonction qui calcul le nombre des mots dans une phrase passée en paramètre

Chapitre 11 : Les chaines des caractères

I. Introduction :

Une chaîne de caractères est une suite ordonnée de caractères.

En algorithmique, une chaîne de caractères est considérée comme un tableau de caractères.

En Python, les chaînes de caractères sont classées dans les séquences tout comme les listes et les tuples **(les tuples et les chaînes sont des séquences immuables, c.à.d. qu'elles ne peuvent pas être modifiées)**.

Et on a vu que pour déclarer une chaine on doit utiliser les guillemets ou les simples quotes. On a vu aussi comment on peut les manipuler en les concaténant ensemble ou les parcourir avec la boucle for, mais en fait avec les chaines (string) on n'arrête pas ici mais il y a plein de méthodes avec lesquelles on peut manipuler les chaines

II. Code ASCII :

Dans un système informatique, à chaque caractère est associé une valeur numérique : son code ASCII (American Standard Code for Information Interchange). L'ensemble des codes est recensé dans une table nommée "table des codes ASCII". Quand on stocke un caractère en mémoire (dans une variable), on mémorise en réalité son code ASCII. Un code ASCII est codé sur un octet (huit bits).

- **Comparaison des caractères :**

Voici un exemple de comparaison de caractères :

"B" > "A" car le code ASCII du caractère "B" (66 en base 10) est supérieur au code ASCII du caractère "A" (65 en base 10).

- Pour comparer deux chaînes de caractères, on compare les caractères de même rang dans les deux chaînes en commençant par le premier caractère de chaque chaîne (le premier caractère de la première chaîne

est comparé au premier caractère de la seconde chaîne, le deuxième caractère de la première chaîne est comparé au deuxième caractère de la seconde chaîne, et ainsi de suite…).

Exemples : Comparaison de deux chaînes

- **"baobab" < "sapin"** car le code ASCII de "b" est inférieur au code ASCII de "s".
- **"baobab" > "banania"** car le code ASCII de "o" est supérieur au code ASCII de "n" (la comparaison ne peut pas se faire sur les deux premiers caractères car ils sont identiques).
- **"1999" > "1998"** car le code ASCII de "9" est supérieur au code ASCII de "8" (la comparaison ne peut pas se faire sur les trois premiers caractères car ils sont identiques). Attention, ici ce ne sont pas des valeurs numériques qui sont comparées, mais bien des caractères.
- **"333" > "1230"** car le code ASCII de "3" est supérieur au code ASCII de "1".
- **"333" < "3330"** car la seconde chaîne a une longueur supérieure à celle de la première (la comparaison ne peut pas se faire sur les trois premiers caractères car ils sont identiques).
- **"Baobab" < "baobab"** car le code ASCII de "b" est supérieur au code ASCII de "B".

III. La manipulation des chaines de caractère en algorithme :

- La fonction **SSCHAINE (chaine , position , nombre) : chaine**

Le rôle de la fonction SSCHAINE() est de retourner une "sous-chaîne" (copie d'un extrait de la chaîne passée en premier paramètre) de nombre caractères de la chaîne chaîne à partir du caractère qui se trouve en position **position**

- La longueur de la chaine : la fonction **LONGUEUR(chaine) :entier**

La fonction LONGUEUR() a pour rôle de calculer et de retourner le nombre de caractères présents dans la chaîne de caractères passée en paramètre.

- La fonction **RANG(chaine , sous-chaine , position) : entier**

La fonction RANG() recherche et retourne la première occurrence de la sous-chaîne dans la chaîne. La recherche commence à partir de position. Cette fonction retourne la position du premier caractère de la sous-chaîne, ou la valeur -1 si la "sous-chaîne" n'existe pas dans la chaîne (à partir de position).

- La fonction **CODE(caractere) : entier**

La fonction CODE() retourne le code ASCII du caractère passé en paramètre.

- La fonction **CAR(code_ascii) :caractere**

La fonction CAR() retourne le caractère dont le code ASCII est passé en paramètre.

- La fonction **MINUSCULE(chaine) : chaine**

Retourne la même chaine passée en paramètre en minuscule

- La fonction **MAJUSCULE(chaine) : chaine**

Retourne la même chaine passée en paramètre en majuscule

IV. La manipulation des chaines de caractère en python :

Les chaînes de caractères peuvent être considérées comme des listes (de caractères) un peu particulières :

```
>>> animaux = "girafe tigre"
>>> animaux
'girafe tigre'
>>> len(animaux)
12
>>> animaux[3]
'a'
```

Nous pouvons donc utiliser certaines propriétés des listes comme les tranches :

```
>>> animaux = "girafe tigre"
>>> animaux[0:4]
'gira'
>>> animaux[9:]
'gre'
>>> animaux[:-2]
'girafe tig'
>>> animaux[1:-2:2]
'iaetg'
```

Mais contrairement aux listes, les chaînes de caractères présentent toutefois une différence notable, ce sont **des listes non modifiables**. Une fois une chaîne de caractères définie, vous ne pouvez plus modifier un de ses éléments. Le cas échéant, Python renvoie un message d'erreur :

```
>>> animaux = "girafe tigre"
>>> animaux[4]
'f'
>>> animaux[4] = "F"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Par conséquent, si vous voulez modifier une chaîne de caractères, vous devez en construire une nouvelle. Pour cela, n'oubliez pas que les opérateurs de concaténation (+) et de duplication (*) (introduits dans le chapitre 2 *Variables*) peuvent vous aider. Vous pouvez également générer une liste, qui elle est modifiable, puis revenir à une chaîne de caractères (voir plus bas).

V. Caractères spéciaux :

Il existe certains caractères spéciaux comme \n que nous avons déjà vu (pour le retour à la ligne). Le caractère \t produit une tabulation. Si vous voulez écrire des guillemets simples ou doubles et que ceux-ci ne soient pas confondus avec les guillemets de déclaration de la chaîne de caractères, vous pouvez utiliser \' ou \".

```
>>> print("Un retour à la ligne\npuis une tabulation\t puis un guillemet\"")
Un retour à la ligne
puis une tabulation     puis un guillemet"
>>> print('J\'affiche un guillemet simple')
J'affiche un guillemet simple
```

Vous pouvez aussi utiliser astucieusement des guillemets doubles ou simples pour déclarer votre chaîne de caractères :

```
>>> print("Un brin d'ADN")
Un brin d'ADN
>>> print('Python est un "super" langage de programmation')
Python est un "super" langage de programmation
```

Quand on souhaite écrire un texte sur plusieurs lignes, il est très commode d'utiliser les guillemets triples qui conservent le formatage (notamment les retours à la ligne) :

```
>>> x = """souris
... chat
... abeille"""
>>> x
'souris\nchat\nabeille'
>>> print(x)
souris
chat
abeille
```

VI. Méthodes associées aux chaînes de caractères

Voici quelques méthodes spécifiques aux objets de type str :

```
>>> x = "girafe"
>>> x.upper()
'GIRAFE'
>>> x
'girafe'
>>> 'TIGRE'.lower()
'tigre'
```

- Les méthodes **.lower()** et **.upper()** renvoient un texte en minuscule et en majuscule respectivement. On remarque que l'utilisation de ces méthodes n'altère pas la chaîne de caractères de départ mais renvoie une chaîne de caractères transformée.

Pour mettre en majuscule la première lettre seulement, vous pouvez faire :

```
>>> x[0].upper() + x[1:]
'Girafe'
```

- Les méthodes **.isupper()** et **.islower()** retournent True si la chaine est toute en majuscule ou minuscule
- La méthode **.capitalize()** plus simplement utiliser la méthode adéquate :

```
>>> x.capitalize()
'Girafe'
```

- la méthode **.split()** Il existe une méthode associée aux chaînes de caractères qui est particulièrement pratique :

```
>>> animaux = "girafe tigre singe souris"
>>> animaux.split()
['girafe', 'tigre', 'singe', 'souris']
>>> for animal in animaux.split():
...     print(animal)
...
girafe
tigre
singe
souris
```

La méthode **.split()** découpe une chaîne de caractères en plusieurs éléments appelés *champs*, en utilisant comme séparateur n'importe quelle combinaison « d'espace(s) blanc(s) ».

Un espace blanc (*whitespace* en anglais) correspond aux caractères qui sont invisibles à l'œil, mais qui occupent de l'espace dans un texte. Les espaces blancs les plus classiques sont l'espace, la tabulation et le retour à la ligne.

Il est possible de modifier le séparateur de champs, par exemple :

```
>>> animaux = "girafe:tigre:singe::souris"
>>> animaux.split(":")
['girafe', 'tigre', 'singe', '', 'souris']
```

Attention, dans cet exemple, le séparateur est un seul caractère « : » (et non pas une combinaison de un ou plusieurs :) conduisant ainsi à une chaîne vide entre singe et souris.

Il est également intéressant d'indiquer à .split() le nombre de fois qu'on souhaite découper la chaîne de caractères avec l'argument maxsplit :

```
>>> animaux = "girafe tigre singe souris"
>>> animaux.split(maxsplit=1)
['girafe', 'tigre singe souris']
>>> animaux.split(maxsplit=2)
['girafe', 'tigre', 'singe souris']
```

- La méthode **.find(sous-chaine , position)**, quant à elle, recherche une chaîne de caractères passée en argument :

```
>>> animal = "girafe"
>>> animal.find("i")
1
>>> animal.find("afe")
3
>>> animal.find("z")
-1
>>> animal.find("tig")
-1
```

Si l'élément recherché est trouvé, alors l'indice du début de l'élément dans la chaîne de caractères est renvoyé. Si l'élément n'est pas trouvé, alors la valeur -1 est renvoyée.

Si l'élément recherché est trouvé plusieurs fois, seul l'indice de la première occurrence est renvoyé :

```
>>> animaux = "girafe tigre"
>>> animaux.find("i")
1
```

- La méthode **.index(sous-chaine , position) :** recherche une sous chaine et renvoie sa position si la sous chaine est trouvé sinon génère une exception (ValueError)

- La fonction **ord(caractere) :** renvoie le code ascii d'un caractère passé en paramètre .
- La fonction **chr(code_ascii)** : renvoie le caractère correspondant au code ascii passé en paramètre
- la méthode **.replace(ancienne_chaine , nouvelle_chaine)** qui substitue une chaîne de caractères par une autre :

```
>>> animaux = "girafe tigre"
>>> animaux.replace("tigre", "singe")
'girafe singe'
>>> animaux.replace("i", "o")
'gorafe togre'
```

- La méthode **.count(chaine)** compte le nombre d'occurrences d'une chaîne de caractères passée en argument :

```
>>> animaux = "girafe tigre"
>>> animaux.count("i")
2
>>> animaux.count("z")
0
>>> animaux.count("tigre")
1
```

- La méthode **.startswith(), .endswith()** vérifie si une chaîne de caractères commence par une autre chaîne de caractères :

```
>>> chaine = "Bonjour monsieur le capitaine !"
>>> chaine.startswith("Bonjour")
True
>>> chaine.startswith("Au revoir")
False
```

Cette méthode est particulièrement utile lorsqu'on lit un fichier et que l'on veut récupérer certaines lignes commençant par un mot-clé. Par exemple dans un fichier PDB, les lignes contenant les coordonnées des atomes commencent par le mot-clé ATOM.

- la méthode **.strip()** permet de « nettoyer les bords » d'une chaîne de caractères :

```
>>> chaine = "  Comment enlever les espaces au début et à la fin ?
>>> chaine.strip()
'Comment enlever les espaces au début et à la fin ?'
```

La méthode **.strip()** enlève les espaces situés sur les bords de la chaîne de caractère mais pas ceux situés entre des caractères visibles. En réalité, cette méthode enlève n'importe quel combinaison « d'espace(s) blanc(s) » sur les bords, par exemple :

```
>>> chaine = "  \tfonctionne avec les tabulations et les retours à la ligne\n"
>>> chaine.strip()
'fonctionne avec les tabulations et les retours à la ligne'
```

La méthode .strip() est très pratique quand on lit un fichier et qu'on veut se débarrasser des retours à la ligne.

VII. Extraction de valeurs numériques d'une chaîne de caractères :

Une tâche courante en Python est de lire une chaîne de caractères (provenant par exemple d'un fichier), d'extraire des valeurs de cette chaîne de caractères puis ensuite de les manipuler.

On considère par exemple la chaîne de caractères val :

```
>>> val = "3.4 17.2 atom"
```

On souhaite extraire les valeurs 3.4 et 17.2 pour ensuite les additionner.

Dans un premier temps, on découpe la chaîne de caractères avec la méthode .split() :

```
>>> val2 = val.split()
>>> val2
['3.4', '17.2', 'atom']
```

On obtient alors une liste de chaînes de caractères. On transforme ensuite les deux premiers éléments de cette liste en *floats* (avec la fonction float()) pour pouvoir les additionner :

```
>>> float(val2[0]) + float(val2[1])
20.599999999999998
```

Remarque : Retenez bien l'utilisation des instructions précédentes pour extraire des valeurs numériques d'une chaîne de caractères. Elles sont régulièrement employées pour analyser des données depuis un fichier.

VIII. Conversion d'une liste de chaînes de caractères en une chaîne de caractères :

On a vu dans le chapitre 2 *Variables* la conversion d'un type simple (entier, *float* et chaîne de caractères) en un autre avec les fonctions int(), float() et str(). La conversion d'une liste de chaînes de caractères en une chaîne de caractères est un peu particulière puisqu'elle fait appelle à la méthode **.join().**

```
>>> seq = ["A", "T", "G", "A", "T"]
>>> seq
['A', 'T', 'G', 'A', 'T']
>>> "-".join(seq)
'A-T-G-A-T'
>>> " ".join(seq)
'A T G A T'
>>> "".join(seq)
'ATGAT'
```

Les éléments de la liste initiale sont concaténés les uns à la suite des autres et intercalés par un séparateur qui peut être n'importe quelle chaîne de caractères. Ici, on a utilisé un tiret, un espace et rien (une chaîne de caractères vide).

Attention, la méthode .join() ne s'applique qu'à une liste de chaînes de caractères.

```
>>> maliste = ["A", 5, "G"]
>>> " ".join(maliste)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 1: expected string, int found
```

On espère qu'après ce petit tour d'horizon vous serez convaincu de la richesse des méthodes associées aux chaînes de caractères. Pour avoir une liste exhaustive de l'ensemble des méthodes associées à une variable particulière, vous pouvez utiliser la fonction dir().

```
>>> animaux = "girafe tigre"
>>> dir(animaux)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '_
_getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '_
_init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mo
d__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__'
, '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'for
mat_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'i
sidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'is
title', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Pour l'instant, vous pouvez ignorer les méthodes qui commencent et qui se terminent par deux tirets bas (*underscores*) __.

Vous pouvez également accéder à l'aide et à la documentation d'une méthode particulière avec help(), par exemple pour la méthode .split() :

```
>>> help(animaux.split)
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator.
(END)
```

Attention à ne pas mettre les parenthèses à la suite du nom de la méthode. L'instruction correcte est help(animaux.split) et non pas help(animaux.split())

IX. Exercices :

Exercice 1 : Ecrivez un algorithme puis un programme en python qui permet de convertir une chaine saisis par l'utilisateur en majuscule

Exercice 2 : Ecrivez un algorithme puis un programme en python qui compte le nombre occurrences d'une lettre donnée dans cette chaîne.

Exercice 3 : Ecrivez un algorithme puis un programme en python qui compte le nombre occurrences des lettres et nombres.

Exercice 4 : Ecrivez un algorithme puis un programme en python qui permet de calculer le nombre de répétition de chaque caractère de la chaine.

Exercice 5 : Ecrivez un algorithme puis un programme en python qui permet de calculer le nombre de voyelles dans une chaine de caractères.

Exercice 6 : Ecrivez un algorithme puis un programme en python qui supprime toutes les lettres « e » (minuscules) d'un texte de moins d'une ligne (supposée ne pas dépasser 132 caractères) fourni au clavier.

Exercice 7 : Ecrivez un algorithme puis un programme en python qui supprime toutes les voyelles d'un texte de moins d'une ligne fourni au clavier.

Exercice 8 : Ecrivez un algorithme puis un programme en python qui permet de supprimer les doublons dans une chaine de caractère saisis par l'utilisateur.

Exercice 9 : Soit la liste ['girafe', 'tigre', 'singe', 'souris'].

1. Affichez chaque élément ainsi que sa taille (nombre de caractères).
2. Affichez pour chaque élément le nombre des voyelles et le nombre des consonnes.