

1 Introduction

Le langage de programmation Python a été créé en 1989 par Guido van Rossum, aux Pays-Bas.

Le nom Python vient d'un hommage à la série télévisée Monty Python's Flying Circus dont G. van Rossum est fan.

La première version publique de ce langage a été publiée en 1991.

La dernière version de Python est la version 3, plus précisément la version 3.8.5 qui a été publiée en juillet 2020 (voir www.python.org).

La Python Software Foundation est l'association qui organise le développement de Python et anime la communauté de développeurs et d'utilisateurs.

Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

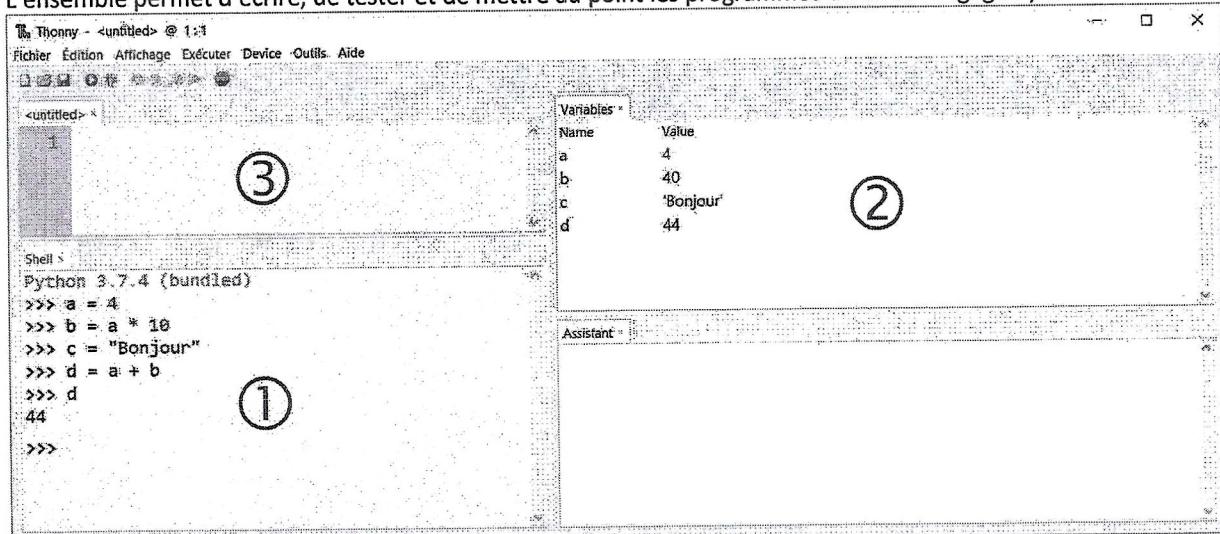
- Il est multiplateforme. C'est-à-dire qu'il fonctionne sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.
- Il est gratuit. Vous pouvez l'installer sur autant d'ordinateurs que vous voulez (même sur votre téléphone !).
- C'est un langage de haut niveau. Il demande relativement peu de connaissance sur le fonctionnement d'un ordinateur pour être utilisé.
- C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.
- Il est orienté objet. C'est-à-dire qu'il est possible de concevoir en Python des entités avec un certain nombre de règles de fonctionnement et d'interactions.
- Il est relativement simple à prendre en main.

Toutes ces caractéristiques font que Python est désormais enseigné dans de nombreuses formations, depuis l'enseignement secondaire jusqu'à l'enseignement supérieur.

2 Distribution Thonny

Thonny est un environnement de développement intégré pour Python conçu pour les débutants. Il comporte un éditeur de programmes, des modules et d'autres utilitaires.

L'ensemble permet d'écrire, de tester et de mettre au point les programmes écrits en langage Python.



① La fenêtre qui porte le nom de **Shell** est la console dans laquelle on peut saisir directement des commandes et obtenir le résultat de l'interprétation de ces commandes.

Les trois signes `>>>` représentent l'invite de commande, c'est donc là que l'on saisit les commandes que l'on souhaite exécuter.

② La fenêtre qui porte le nom de **Variables** permet de visualiser la valeur affectée aux variables utilisées.

③ Cette fenêtre permet de saisir plusieurs commandes successives, c'est-à-dire un **script** que l'on exécutera en une fois.

3 Utilisation de la console

Un shell est un interpréteur de commandes permettant d'interagir avec l'ordinateur. On utilisera le shell pour exécuter un script très court en Python. Un shell possède toujours une invite de commande, c'est-à-dire un « symbole » qui s'affiche avant l'endroit où on entre des commandes. Ici, cette invite est représentée par le symbole `>>>`

Par exemple, si on vous demande de lancer l'instruction suivante : `>>> print("Bonjour")`

Il faudra taper seulement `print("Bonjour")` sans le `>>>` ni l'espace après le `>>>`, puis actionner la touche « Entrée » pour valider la commande et l'exécuter.

La console permet aussi de visualiser le résultat de la commande ou du script saisi.

```

Shell>
>>> print("Bonjour")
Bonjour
>>>

```

Dans la console, saisir les commandes ci-dessous, indiquer le résultat obtenu, conclure sur l'opération effectuée :

Commande à saisir	Résultat obtenu	Opération réalisée
<code>>>> 3+2</code>	5	addition
<code>>>> 7-4</code>	3	soustraction
<code>>>> 3*4</code>	12	multiplication
<code>>>> 23/3</code>	7,666	division arrondie par défaut
<code>>>> 23//3</code>	7	division euclidienne
<code>>>> 23%3</code>	2	donne le reste de la division
<code>>>> 2**4</code>	16	-puissance
<code>>>> 7+3*4</code>	19	multiplication de 3 par 4
<code>>>> (7+3)*4</code>	40	addition de 7 + 3 multiplié par 4
<code>>>> abs(-3.2)</code>	3,2	met la valeur absolue et enlève le signe
<code>>>> round(3.44)</code>	3	arrondir par excès
<code>>>> round(3.54)</code>	4	arrondir par excès
<code>>>> round(3.57,1)</code>	3,6	arrondir avec une décimale
<code>>>> round(23/3,5)</code>	7	arrondir une division avec 5 décimales
<code>>>> 1000*1.2e-3</code>	1,2	.
<code>>>> pow(2,10)</code>	1024	2 exposant 10

Les lignes ci-dessous utilisent des variables a et b auxquelles on affecte une valeur qui reste en mémoire

<code>>>> a=5</code>	15	Affectation d'une variable
<code>>>> 3*a</code>	45	.
<code>>>> b=4</code>	19	affectation et addi. d'une variable
<code>>>> a+b</code>	23	.
<code>>>> 4*a+5*b</code>	90	multiplication d'a et b puis addition de ?
<code>>>> a,b=4.8,12.7</code>	17.5	Affectation simultanée de deux variables
<code>>>> a+b</code>	17.5	l'écriture et additionner a+b
<code>>>> print("a + b = ",a+b)</code>	a + b = 17.5	.

4 Utilisation de l'éditeur

Lorsque l'on doit écrire plusieurs commandes successives, un programme par exemple, on a recours à l'éditeur qui permet en outre de sauvegarder l'ensemble de la saisie.

Créer un nouveau fichier : « Fichier » « Nouveau » puis « Fichier » « Enregistrer »
Le nommer **TestParite.py** en le sauvegardant dans un dossier **h:\NSI\Python** que vous aurez créé préalablement.

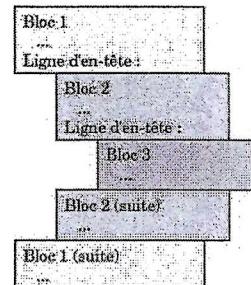
Saisir l'intégralité du programme proposé ci-dessous. *Les commentaires ne sont pas en option !!!*

```
TestParite.py
1 # Le programme demande de saisir un nombre et il indique s'il est pair ou impair
2
3 nombre = int(input("Entrez un nombre entier compris entre 0 et 100 : ")) # transforme le texte saisi au clavier en un nombre
4 if nombre%2 == 0:
5     print("Le nombre",nombre,"est pair")
6 else:
7     print("Le nombre",nombre,"est impair")
```

L'instruction **if** impose un bloc d'indentation : la ligne d'instruction qui suit est décalée.

Il peut y avoir plusieurs indentations, ce qui permet de structurer le programme.

Le schéma ci-contre en résume le principe.



Tester et valider le fonctionnement de votre programme avec plusieurs valeurs numériques.

Que se passe-t-il si l'on saisit du texte ? *Le programme me demande alors*

qui correspond à un nombre et donc ça crée une erreur.

input()

Demander une valeur à l'utilisateur et l'affecter à une variable a :

Toute entrée donnée avec l'instruction **input** est une chaîne de caractères :

Si on veut que a contienne un nombre entier, on utilise

Si on veut que a contienne un nombre réel (flottant), on utilise

Si on veut que a contienne un nombre sans préciser le type, on utilise

a = input("texte") où "texte" est facultatif

a = int(input("texte"))

a = float(input("texte"))

a = eval(input("texte"))

if ... else

if condition:

 instruction1 si condition vérifiée (une instruction par ligne)

 instruction2 si condition vérifiée

else:

 instruction3 si condition non vérifiée (une instruction par ligne)

Ne pas oublier les « : » à la fin des lignes **if** et **else**

L'indentation est obligatoire, le retour au bord indique la sortie du **if** ou du **else**

Test d'égalité : **a==5**

Test de non égalité : **a!=5**

Tests de comparaison : **a<5 a<=5 a>5 a>=5**

Plusieurs conditions possibles avec les opérateurs booléens **or** (ou) et **and** (et) : **a>5 and a<10**

Exercice 1 :

Écrire un programme **TestAge.py** qui vous demande votre prénom et votre âge puis qui vous affiche un message pour vous saluer et vous indiquer si vous êtes mineur ou majeur.

Ajouter ensuite une phrase qui vous indique depuis combien d'années vous êtes majeur ou combien d'années attendre avant de le devenir.

5 Écriture de scripts complets avec l'éditeur

Il faut prendre de bonnes habitudes de structuration des scripts dès le départ : avec Python, nous allons créer des fonctions qui permettent l'exécution d'une ou plusieurs tâches. Les fonctions sont définies au début du fichier .py et on les appellera à partir d'un programme principal (main) qui sera défini plus bas dans le fichier.

La structure d'un programme complet ressemblera à quelque chose comme ça :

```
ExempleScript.py
1 # Importation d'outils contenus dans une bibliothèque
2 from bibliothèque import *
3
4 def fonction1():
5     """ Documentation sur la fonction 1 """
6     ...
7     return
8
9 def fonction2(argument1,argument2):
10    """ Documentation sur la fonction 2 """
11    ...
12    ...
13    return variable
14
15 # Programme principal : c'est là que l'on définit l'ensemble du traitement qui doit être effectué
16 def main():
17    """ Documentation sur la fonction main """
18    ...
19    fonction1()
20    ...
21    Z = fonction2(X,Y)
22    ...
23
24 # C'est ici que sera appelé le programme principal
25 if __name__ == '__main__':
26    main()
```

Explications sur la fonction main :

L'astuce consiste à utiliser la variable `__name__` qui est une variable automatiquement créée par Python, toujours disponible, n'importe où, dans tous les scripts. Cette variable contient le nom du script courant. Ainsi, dans n'importe quel code, on peut vérifier où l'on est. C'est utile lorsque l'on importe d'autres scripts dans le script que l'on écrit (comme à la ligne 2).

Si le script est le script principal, alors `__name__` ne contient pas le nom du script, il contient la chaîne `__main__`.
Les lignes 25 et 26 disent : Si ce code est exécuté en tant que script principal, appelé directement avec Python et pas importé, alors exécuter cette fonction `main`.

Exercice 2 :

Écrire, en respectant la structure présentée ci-dessus un programme `Maximum.py` qui vous demande deux nombres puis qui vous affiche un message pour vous indiquer lequel est le plus grand des deux.

```
Shell >
>>> %Run Maximum.py
Veuillez saisir un premier nombre SVP : 12.4568
Veuillez saisir un second nombre SVP : 75.1256
Le plus grand nombre saisi est 75.1256
>>>
```

On utilisera une fonction `plus_grand(nombre1,nombre2)` qui sera appelée dans la fonction `main`

6 Les types de variables

Avec Python, le programmeur ne déclare pas le type des variables à utiliser. La déclaration peut se faire implicitement lorsque l'on affecte une valeur à la variable. Les quatre types de base sont les suivants :

- bool booléen (True ou False)
- int nombre entier
- float nombre réel ou flottant
- str chaîne de caractères (string)

Si un traitement nécessite un changement de type, Python peut l'effectuer dans certaines mesures (parfois, une erreur se produit).

À tout moment, on peut obtenir le type d'une variable var en saisissant dans la console `type(var)`

Saisir dans la console les commandes suivantes, consigner le résultat obtenu et conclure.

Commande à saisir	Résultat obtenu	Explication ou conclusion
<code>>>> a=3 >>> type(a)</code>	<code>< class 'int' ></code>	La variable a est un entier
<code>>>> b=3.14 >>> type(b)</code>	<code>< class 'float' ></code>	$b \rightarrow$ décimal
<code>>>> c=(a<b) >>> c >>> type(c)</code>	<code>< class 'bool' ></code>	$c \rightarrow$ c'est soit true soit false
<code>>>> c=(a>b) >>> c >>> type(c)</code>	<code>< class 'bool' ></code>	$c \rightarrow$ f
<code>>>> d="bonjour" >>> type(d) >>> d=d*2 >>> d</code>	<code>< class 'str' ></code>	$d \rightarrow$ c'est un mot
<code>>>> a=3 >>> type(a) >>> a=a+0.1 >>> type(a)</code>	<code>< class 'int' > < class 'float' ></code>	$a \rightarrow$ se transforme en float car deviens décimal
<code>>>> b=3.14 >>> type(b) >>> b=int(b) >>> b >>> type(b)</code>	<code>< class 'float' > < class 'int' ></code>	on a transformé b en entier ($3,14 \rightarrow 3$)
<code>>>> a=3 >>> type(a) >>> a=float(a) >>> a >>> type(a)</code>	<code>< class 'int' > < class 'float' ></code>	a prend le type float \rightarrow il peut être décimal
<code>>>> d=6.35 >>> type(d) >>> s="Valeur de d = "+repr(d) >>> s >>> type(s)</code>	<code>< class 'float' > < class 'str' ></code>	assimile une phrase à d
<code>>>> s="1024" >>> type(s) >>> a=int(s) >>> a >>> type(a)</code>	<code>< class 'str' > < class 'int' ></code>	a prend la valeur de s en type int
<code>>>> a=3 >>> b=4 >>> c=(a==b) >>> c >>> type(c) >>> b=b-1 >>> c=(a==b) >>> c >>> c=c*2 >>> type(c)</code>	<code>< class 'int' > < class 'int' ></code>	on donne des valeurs à a et b et on fait vérifier si elles sont égales avec c

Exercice 3 :

Écrire dans l'éditeur un script **Calculs.py** de six lignes ... :

1. qui affecte un nombre à une variable x
2. qui ajoute 7 au triple du nombre donné
3. qui multiplie le résultat par le nombre donné
4. qui soustrait au résultat le nombre donné
5. qui affiche le résultat obtenu
6. qui affiche le type du résultat obtenu

$x = \text{float}(\text{input}("Choisis un nombre"))$
 $x = x \times 3 + 7$
 $x = x - x$
 $\text{print}(x)$
 $\text{print}(\text{type}(x))$

Conseils : Créer le minimum de variables. Tester avec un nombre entier, puis avec un nombre réel.

x = 5 y = 105

x = 5,1 y = 108,629999

Calculer $y = f(x)$ pour que l'équation tienne sur une seule ligne

Ajouter à la suite de votre script, dans le **main**, la ligne ci-contre : et une seconde ligne qui affiche la valeur de z.

Exécuter le script pour $x=5.1$, que constatez-vous ?

$z = 3 \times x \times x^2 + 6 \times x$ # fonction obtenue pour y

Exercice 4 :

Écrire dans l'éditeur un script **Pesanteur.py** qui vous demande à quelle altitude h vous êtes (en mètres) puis qui calcule et affiche dans une phrase l'intensité de la pesanteur g avec son unité.

On rappelle que $g = \frac{G \times m_{Terre}}{(R_{Terre} + h)^2}$ avec $R_{Terre} = 6378 \text{ km}$ $G = 6,674 \times 10^{-11} \text{ USI}$ $m_{Terre} = 6,0 \times 10^{24} \text{ kg}$
(G est la constante universelle de gravitation)

```
Shell > 
>>> %Run Pesanteur.py
Quelle est l'altitude, en mètres, à laquelle vous vous trouvez ? 67
La valeur de g à l'altitude de 67 mètres est 9,844 m/s^2
>>>
```

Conseil : pour obtenir un format d'affichage lisible de la valeur de g, vous utiliserez la commande adéquate.

def main():

""" Calcul de la pesanteur en fonction de l'altitude """

altitude =

R_Terre =

Exercice 5 :

Écrire dans l'éditeur un script **Conversions.py**

- qui vous demande la base d'origine du nombre à convertir (2 ou 16)
- qui vous demande un nombre binaire ou hexadécimal à convertir
- qui effectue la conversion en base 10 de ce nombre

Vous écrirez deux fonctions **TwoToTen(nombreBinaire)** et **HexToTen(nombreHexadecimal)**. Vous appellerez l'une ou l'autre selon le choix fait par l'utilisateur.

Avant d'écrire le script, essayer dans la console les commandes proposées ci-dessous :

```
>>> int("10100000", base=2) ..... 160  
>>> int("F0", base=16) ..... 240
```

La structure du script est donnée ci-dessous, complétez les six lignes manquantes. Saisir le script et valider le fonctionnement.

```
def TwoToTen(nombreBinaire):  
    """ convertit la chaîne nombreBinaire en un nombre en base 10 """  
    ..... int(..... nombreBinaire, base = 2 )  
    return nombreDecimal  
  
def HexToTen(nombreHexadecimal):  
    """ convertit la chaîne nombreHexadecimal en un nombre en base 10 """  
    ..... int(..... nombreHexadecimal, base = 16 )  
    return nombreDecimal  
  
def main():  
    """ Demande la base d'origine du nombre à convertir (2 ou 16)  
    Demande le nombre binaire ou hexadécimal à convertir  
    Effectue la conversion en base 10 du nombre saisi """  
    maBase = ..... input('Quelle est la base?')  
    nombre = ..... input('Quelle est le nombre à convertir?')  
    if maBase == 2:  
        resultat=TwoToTen(nombre) # appel de la fonction de conversion  
        ..... print('Le nombre est .....')  
    elif maBase==16:  
        resultat=HexToTen(nombre) # appel de la fonction de conversion  
        ..... return resultat  
    else:  
        print("Tu dois choisir uniquement la base 2 ou la base 16 !")  
  
if __name__=='__main__':  
    main()
```

Question : Aurions-nous pu simplifier le script en utilisant une seule fonction de conversion ?

— Oui, on n'avait qu'à mettre les deux fonctions en une —

7 Opérations sur les chaînes de caractères

Les types **bool**, **int** et **float** sont des types simples. Le type **str** est un type simple particulier, chaque caractère de la chaîne a un indice propre qui permet d'accéder directement à sa valeur (voir exemples ci-dessous).

Avec Python, le programmeur déclare une chaîne de caractères entre guillemets ou entre deux apostrophes.

Python est sensible à la casse, il distingue les minuscules et les majuscules.

Commande à saisir	Résultat obtenu	Explication ou conclusion
<code>>>> prenom='Jean'</code>	'Jean'	Affectation d'une variable de type chaîne
<code>>>> prenom</code>		
<code>>>> nom = "Dupont"</code>	'Dupont'	Affectation d'une variable de type chaîne
<code>>>> nom</code>		
<code>>>> prenom+nom</code>	'JeanDupont'	+
<code>>>> print(prenom,nom)</code>	Jean Dupont	écrit les variables nom et prenom (espace auto avec print)
<code>>>> print(prenom+nom)</code>	'JeanDupont'	écrit les variables mises attaqué
<code>>>> print(prenom+' '+nom)</code>	'Jean Dupont'	ajoute un espace entre les deux variables
<code>>>> len(nom)</code>	6	len() donne la longueur de la variable
<code>>>> phrase="Bonjour, où se trouve l'université ?"</code>	'Bonjour ...'	Affectation d'une variable de type chaîne
<code>>>> phrase[0]</code>	'B'	donne la première lettre
<code>>>> phrase[1]</code>	'o'	donne la 2 ^e lettre
<code>>>> phrase[3:7]</code>	'jour'	donne de la 4 ^e à la 7 ^e lettre
<code>>>> phrase[-1]</code>	'?'	donne la dernière lettre
<code>>>> phrase[-12:]</code>	'iniversité ?'	de la 12 ^e lettre en partant de la fin jusqu'au bout de la phrase
<code>>>> phrase[9:]</code>	' où se trouve l'université ?'	donne de la 10 ^e à la fin de la phrase
<code>>>> date='Aujourd\'hui'</code>	'Aujourd'hui'	\ est le caractère d'échappement
<code>>>> date</code>		
<code>>>> nom.lower()</code>	'jean'	s.lower() est la méthode qui place tous les caractères de la chaîne s en minuscule
<code>>>> nom.upper()</code>	'DU PONT'	s.upper() est la méthode qui place tous les caractères de la chaîne s en majuscule
<code>>>> nom.center(12,'-')</code>	'--- Dupont ---' 3 6 3	s.center(n,car) est la méthode qui permet de centrer la variable = 12
<code>>>> phrase='Bonjour, où se trouve Ouagadougou ?'</code>	'Bonjour ...'	Affectation d'une variable de type chaîne
<code>>>> phrase</code>		
<code>>>> phrase.count('ou')</code>	9	s.count(sub) est la méthode qui permet de compter les syllabes demandées
<code>>>> phrase.replace('o','i')</code>	'Binjvi, ii de l'veuve Lingadingui ?'	s.replace(old,new) est la méthode qui permet de remplacer une lettre par une autre

8 Structures itératives : les boucles

Lorsque l'on doit répéter un traitement plusieurs fois, on a recours à une structure algorithmique appelée « boucle ».

8.1 LES BOUCLES CONDITIONNELLES

La structure est :

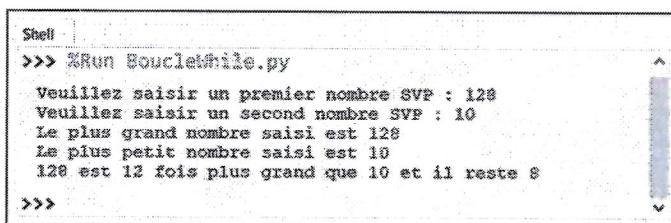
```
while condition :
    instruction1
    instruction2
    ...
    ...
```

La condition est une expression dont la valeur interprétée est **True** ou **False** → *bouclée*

Tant que la condition est vérifiée (True), les instructions sont exécutées. Au moins l'une de ces instructions modifie une variable qui va amener la condition à ne plus être vérifiée (False) après un nombre d'itérations sinon on ne sort plus de la boucle.

Exercice 6 :

Saisir et compléter le script **BoucleWhile.py** donné ci-dessous, on veut obtenir, après saisi de deux entiers l'affichage suivant :



```
Shell >>> %run BoucleWhile.py
Veuillez saisir un premier nombre SVP : 128
Veuillez saisir un second nombre SVP : 10
Le plus grand nombre saisi est 128
Le plus petit nombre saisi est 10
128 est 12 fois plus grand que 10 et il reste 8
>>>
```

```
def plus_grand(nombre1,nombre2):
    """ détermine quel est le plus grand nombre et l'affecte à la variable max
    détermine quel est le plus petit nombre et l'affecte à la variable min"""
    if nombre1 > nombre2:
        maxi = nombre1
        mini = nombre2
    else:
        maxi = nombre2
        mini = nombre1
    return maxi,mini

def main():
    """ Demande de saisir deux nombres entiers
    Affiche le plus grand et le plus petit
    Affiche le nombre de fois que l'un contient l'autre et le reste"""
    q = 0
    x = int(input("Veuillez saisir un ..."))
    y = int(input("Veuillez saisir un second ..."))
    a,b = plus_grand(x,y)
    print("Le plus grand nombre est",a)
    print("Le plus petit nombre est",b)
    a_initial = a

    while a >= b:
        a = a - b
        q = q + 1

    print(a - initial,"est",q,"fois plus grand que",b,"et il reste",a)
    # affiche les résultats

if __name__=='__main__':
    main()
```

Annotations manuscrites sur le code :

- maxi = *nombre1*
- mini = *nombre2*
- maxi = *nombre2*
- mini = *nombre1*
- # initialisation du quotient
- # demande un premier entier
- # demande un second entier
- # appel de la fonction
- # affiche le plus grand des deux nombres
- # affiche le plus petit des deux nombres saisis
- # sauvegarde a avant les traitements
- # si a est plus grand que b ou égal à b
- # on retranche b à a
- # incrémenter le quotient de 1
- # affiche les résultats

8.2 LES BOUCLES NON CONDITIONNELLES

Une boucle non conditionnelle permet de répéter un nombre de fois connu un bloc d'instructions.

La structure est :

```
for i in range(n):
    instruction1
    instruction2
    ...
    ...
```

La variable *i* est créée, elle va prendre successivement les valeurs 0, 1, 2, 3, ..., n-1
Les instructions seront donc exécutées n fois.

Exercice : Tester avec la console ou avec un petit script les instructions suivantes, consigner les résultats obtenus :

instruction	Séquence des nombres obtenus
range(5)	0 1 2 3 4
range(3,8)	3 4 5 6 7
range(2,12,3)	2 5 8 11
range(20,5,-5)	20 15 10

de 0 à 5 exclu
de 3 à 8 exclu
de 2 à 12 mais 3 par 3
de 20 à 5 exclu mais 5 par 5

La syntaxe complète est **range(d,f,p)** qui renvoie la séquence des entiers de d inclus à f exclus avec un pas égal à p
Avec $d < f$ et $p > 0$ ou bien $d > f$ et $p < 0$

par défaut $d = 0$ et $p = 1$

Remarque : La boucle for peut être utilisée avec une liste ou une chaîne de caractères.

Tester le script suivant

```
>>> for lettre in "Bonjour":
        print(25 * lettre)
```

```
Shell
>>> for lettre in "Bonjour":
        print(25 * lettre)

BBBBBBBBBBBBBBBBBBBBBBBBBBBB
ooooooooooooooooooooooooooo
nnnnnnnnnnnnnnnnnnnnnnnnnn
jjjjjjjjjjjjjjjjjjjjjjjjjj
ooooooooooooooooooooooooooo
uuuuuuuuuuuuuuuuuuuuuuuuuu
rrrrrrrrrrrrrrrrrrrrrrrrrr
```

```
2 puissance 0 = 1
2 puissance 1 = 2
2 puissance 2 = 4
2 puissance 3 = 8
2 puissance 4 = 16
2 puissance 5 = 32
2 puissance 6 = 64
2 puissance 7 = 128
2 puissance 8 = 256
```

Exercice 7 :

Écrire dans l'éditeur un script **BoucleFor.py**

- qui permet d'afficher la valeur des puissances de 2 de 2^0 à 2^{20}
- Vous présenterez les résultats sous la forme présentée ci-contre :

*for i in range (21):
....print ("2 puissance ", i, " = ", 2**i)*

Python supporte le formatage de valeurs en chaînes de caractères, l'usage le plus simple consiste à insérer des valeurs dans des chaînes à l'aide de marques %s. Modifier votre programme pour utiliser la ligne **print()** proposée ci-dessous :

*for i in range (21)
....puissance = 2 ** i
....print ("2 puissance %s = %s" %(n,puissance))*

```
2 puissance 8 = 256
2 puissance 9 = 512
2 puissance 10 = 1024
2 puissance 11 = 2048
2 puissance 12 = 4096
```

Pour aller plus loin : Utiliser la fonction **len()** pour formater l'affichage des valeurs et les aligner verticalement comme présenté ci-contre :

Vous devrez insérer un nombre d'espaces lié à la longueur des nombres n et 2^n

```
2 puissance 15 = 32768
2 puissance 16 = 65536
2 puissance 17 = 131072
2 puissance 18 = 262144
2 puissance 19 = 524288
2 puissance 20 = 1048576
```