

Cours + 300 exercices avec solutions

Les briques de base en langage Python

Ouvrage destiné aux étudiants désireux
d'apprendre à programmer en langage Python en toute simplicité

**PRÉSENTÉ PAR
YOUNES DERFOUFI**

**DCTEUR AGRÉGÉ
FORMATEUR AU
CRMEF OUJDA**

Les briques de base en langage Python

Younes Derfoufi. Docteur Agrégé
Enseignant d'informatiques et de mathématiques
Formateur des enseignants stagiaires
au CRMEF Oujda

4 décembre 2022

Table des matières

I	Les Bases En Python	7
1	Les éléments de base en Python	8
1.1	Installation des outils de développement en Python	10
1.2	Premier programme en Python	14
1.3	Les variables, commentaires & opérateurs en Python	17
1.4	Les fonctions en Python	21
1.5	Structures de contrôles	22
1.6	Les chaînes de caractères en Python	25
1.7	Les listes en Python	31
1.8	Les tuples	36
1.9	Les tableaux (array)	38
1.10	Les dictionnaires	40
1.11	Les ensembles Python (Python sets)	44
1.12	Fonction Lambda En Python	47
1.13	Compréhension des listes en Python	49
2	Programmation orientée objet POO en Python	50
2.1	Le concept de POO en Python	50
2.2	Terminologie de la POO	51
2.3	Les classes en Python	52
2.4	Les méthodes d'instances en Python	53
2.5	Les méthodes de classes en Python	53
2.6	Attributs d'instances et attributs de classes	53
2.7	Les méthodes statiques	54
2.8	Héritage en Python	54
2.9	Héritage multiple	56
2.10	Surcharge de méthodes (overloading)	56
2.11	Polymorphisme et redéfinition de méthodes (overriding methods)	58

2.12	Les classes abstraites en Python	59
2.13	Les interfaces en Python	60
3	Les modules en Python	61
3.1	Introduction	61
3.2	Créer votre propre module	61
3.3	les modules standards en Python	62
4	Exemple d'usage de quelques modules standards	64
4.1	Le module os	64
4.2	Le module statistics	64
4.3	le module virtualenv	66
4.4	Le module PyInstaller : Transformer un script Python en un exécutable Windows	66
4.5	Le module math	67
4.6	Le module random	69
4.7	Le module collection	71
5	Les fichiers en Python	72
5.1	Le module os	72
5.2	Mode d'ouverture d'un fichier	76
5.3	Ouverture et lecture d'un fichier	76
5.4	Lecture et écriture à une position donnée à l'aide de la méthode seek()	80
5.5	Ouverture en mode écriture des fichiers en Python	81
5.6	Récapitulatif des méthodes Python associées à un objet fichier avec description :	83
5.7	Manipulation des fichiers de configuration en Python	83
6	Python et les bases de données SQLite	89
6.1	A propos des bases de données SQLite3	89
6.2	Création de tables et de bases de données SQLite3	89
6.3	Insertion de données	90
6.4	Insertion des données de variables dans une table SQLite	91
6.5	Affichage des données d'une table SQLite3	92
6.6	Mise à jour des données SQLite3	92
6.7	Récupération totale des données sous forme d'un tableau à deux dimension	93
6.8	Exportation du contenu de la base SQLite3 vers une base sql (SQLite3 dump)	93
6.9	Éditeur WYSIWYG SQLite3	94

II	Interfaces Graphiques	96
7	Les bibliothèques d'interfaces graphiques	97
7.1	Première fenêtre graphique avec Tkinter	97
7.2	Les widgets Tkinter	98
7.3	Les attributs standard des widgets Tkinter	119
7.4	Les méthodes de gestion des dispositions géométriques des widgets	120
7.5	Actions manipulant des widgets Tkinter	123
7.6	Menu Tkinter en Python	123
7.7	Les événements en Tkinter (binding event)	126
7.8	La bibliothèque d'images Pillow	131
7.9	Les boîtes de dialogues en Tkinter	134
7.10	Le module de design tkinter.ttk	140
7.11	Obtenir des informations sur la ligne sélectionnée	149
8	Minis Projets En PythonTkinter	151
8.1	Mini Projet : Calculatrice En Python Tkinter	151
8.2	Mini projet : création d'un éditeur de texte	155
8.3	Mini Projet : logiciel de traduction	162
III	Exercices Avec Solutions	168
1.	Exercices sur les bases en python : strings variables	169
2.	Exercices sur les listes Python	227
3.	Exercices sur les algorithmes design Python	283
4.	Exercices d'arithmétiques en Python	288
5.	Exercices sur les dictionnaires Python	313
5.	Exercices sur les ensembles Python	324
6.	Exercices sur les fichiers Python	333
7.	Exercices sur la programmation orientée objet en Python	345
8.	Exercices sur la bibliothèque graphique Tkinter	353

A propos de l'auteur

Younes Derfoufi est un formateur au CRMEF OUJDA, docteur agrégé a plus de 30 ans d'expérience en enseignement des mathématiques et informatiques au élèves professeurs au CRMF Oujda. Il a enseigné : **une dizaine de modules informatiques au CRMEF Oujda : développement web** front end et backend : HTML, CSS, Bootstrap, Javascript, JQuery, PHP, MySql..., **développement web orienté CMS** : Joomla, Wordpress, Moodle, Simple Machine, PHPBB ...Sans oublier les frameworks web : Laravel, Django Et aussi la programmation orientée objet en Java, Dart & FlutterIl a enseigné aussi la programmation Python : les bases en python, programmation orientée objet, bases de données SQLite... Interfaces graphique Python Tkinter & PyQt5... Il a enseigné aussi durant une dizaine d'années les technologie de l'information et de la communication : TICE Multimédia : Autoplay Media Studio, Montage vidéo avec AVS Eideo Editor & Camtasia Studio, traitement d'image avec Gimp... TICE Mathématique : Geogebra, algorithmique avec Algobox, langage LaTeX, Scientific Workplace, Statistique avec SPSS, Statistiques avec Python Numpy & Pandas... Et avant d'enseigné tout ceux-ci, l'auteur a travaillé comme enseignant des mathématiques aux enseignants stagiaires pendant une vingtaine d'années, il enseigné en particulier les notions suivante : **algèbres générale** : groupes anneaux, corps, polynômes à une ou plusieurs indéterminées, **algèbres linéaires** : espaces vectoriels, matrices, forme bilinéaire, forme quadratiques espaces euclidiens et hermitiens, formes multilinéaires alternées & déterminants... séries numériques, suites & séries de fonctions, séries entières fonctions analytiques..., intégration au sens de Riemann, intégrale généralisée, dépendant d'un paramètre, intégrales multiples..., géométrie affines, géométrie vectorielle euclidienne, géométrie affine euclidiennes, probabilités & statistiques... Malgré tout cela, l'auteur n'a eu aucune peine à enseigner la didactiques des mathématiques et informatiques, pédagogie par objectifs PPO, méthodologie des compétences, planification des apprentissages des mathématiques et informatiques, pro-

duction didactique, TICE mathématique et informatiques. L'auteur a aussi participé à de nombreux colloques & séminaires internationaux :

1. séminaires mensuels de topologies robotiques : <https://www.algtop.net/2012/02/14/seminaire/>
2. séminaires sur les compétences de haut niveaux qui s'est tenu au CRMEF de Meknès en 2004
3. Participation au projet genie organisé en 2006, 2008 et 2009 au INPT à Rabat

Droits d’auteur

Conformément à la loi du 11 mars, il est interdit de reproduire intégralement ou partiellement le contenu du présent ouvrage, sur quelque support que ce soit, sans autorisation de l’éditeur ou de ses ayants droit. Une distribution ou dépôt électronique sur n’importe quelle plate forme constitue une contrefaçon sanctionnée par les articles L. 335-2 et du code de la propriété intellectuelle.

Première partie

Les Bases En Python

Chapitre 1

Les éléments de base en Python



1.0.1 Introduction

Python est un langage de programmation de haut niveau interprété pour la programmation à usage général. Créé par **Guido van Rossum**, et publié pour la première fois en 1991. Python repose sur une philosophie de conception qui met l'accent sur la lisibilité du code, notamment en

utilisant des espaces significatifs. Il fournit des constructions permettant une programmation claire à petite et grande échelle.

Python propose un système de typage dynamique et une gestion automatique de la mémoire. Il prend en charge plusieurs paradigmes de programmation, notamment **orienté objet**, **impératif**, **fonctionnel** et **procédural**, et dispose d'une bibliothèque standard étendue et complète.

Python est un langage de programmation open-source et de haut niveau, développé pour une utilisation avec une large gamme de systèmes d'exploitation. Il est qualifié de langage de programmation le plus puissant en raison de sa nature dynamique et diversifiée. Python est facile à utiliser avec une syntaxe super simple très encourageante pour les apprenants débutants, et très motivante pour les utilisateurs chevronnés.

1.0.2 Quelles sont les principales raisons qui poussent à apprendre Python ?

1. **Python est utilisé par des sites web pionniers** : tels que Microsoft, YouTube, Drop Box,... Python a une forte demande sur le marché.
2. **Richesse en outils** : de nombreux IDE sont dédiés au langage Python : **Pycharm**, **Wing**, **PyScripter**, **Spyder**...
3. **Python est orienté objet** : la puissance du langage python est fortement marquée par son aspect orienté objet, qui permet la création et la réutilisation de codes. En raison de cette possibilité de réutilisation, le travail est effectué efficacement et réduit beaucoup de temps. Au cours des dernières années, la programmation orientée objet s'est rapporté non seulement à des classes et des objets, mais à de nombreuses bibliothèques et frameworks. Python à son tour a connu dans ce contexte un grand essor : des **dizaines de milliers de bibliothèques** sont disponibles à l'aide de l'outil **pip** de **gestion des packages**.
4. **Simplicité et lisibilité du code** : Python a une syntaxe simple qui le rend approprié pour apprendre la programmation en tant que premier langage. L'apprentissage est plus fluide et rapide que d'autres langages tels que **Java**, qui nécessite très tôt une connaissance de la programmation orientée objet ou du **C/C++** qui nécessite de comprendre les pointeurs. Néanmoins, il est possible d'en apprendre davantage sur la programmation orientée objet en Python lorsqu'il est temps. Par conséquent, Python peut être utilisé comme prototype et peut être implémenté dans un autre langage de programmation après avoir testé le code.

5. **Python est open source donc gratuit** : Python étant un langage de programmation open source, il est gratuit et permet une utilisation illimitée. Avec cette licence open-source, il peut être modifié, redistribué et utilisée commercialement... Avec cette licence, Python est devenu robuste, doté de capacités évolutives et portables et est devenu un langage de programmation largement utilisé.
6. **Python est multi plateforme** : Python peut être exécuté sur tous les principaux systèmes d'exploitations, tels que : **Mac OS, Microsoft Windows, Linus et Unix...** Ce langage de programmation offre une meilleure expérience de travail avec n'importe quel système d'exploitation.
7. **Python est très puissant en terme de production** : la puissance du langage Python a été démontré sur le terrain du développement :
 - Développement Web, en utilisant les frameworks Django, Flask, Pylons
 - Science des données et visualisation à l'aide de Numpy, Pandas et Matplotlib
 - Applications de bureau avec Tkinter, PyQt, Gtk, wxWidgets et bien d'autres..
 - Applications mobiles utilisant Kivy ou BeeWare
 - Éducation : Python est un excellent langage pour apprendre l'algorithme et la programmation! Par conséquent largement utilisé aux Lycées, Classes préparatoires, Instituts supérieurs, Universités...

1.1 Installation des outils de développement en Python

Afin de pouvoir développer en langage Python, nous devons installer les outils nécessaires :

i) - **Télécharger et installer Python** :

<https://www.python.org/downloads/>

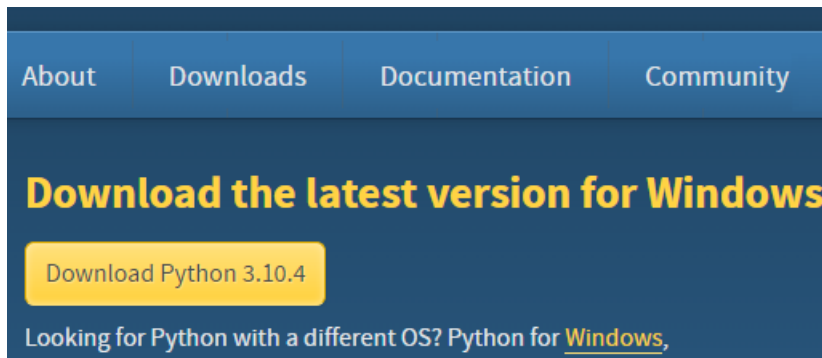
ii) - **Télécharger et installer un IDE Python** : de nombreux choix s'offre à vous : **Pycharm, PyScripter, Wing**. Quant à moi je vous recommande **wing**, en raison de sa rapidité et de sa simplicité d'usage, en plus il est gratuit : Télécharger Wing :

<https://wingware.com/downloads/wing-personal>

1.1.1 Installation de Python

Nous traitons ici le cas de Windows :

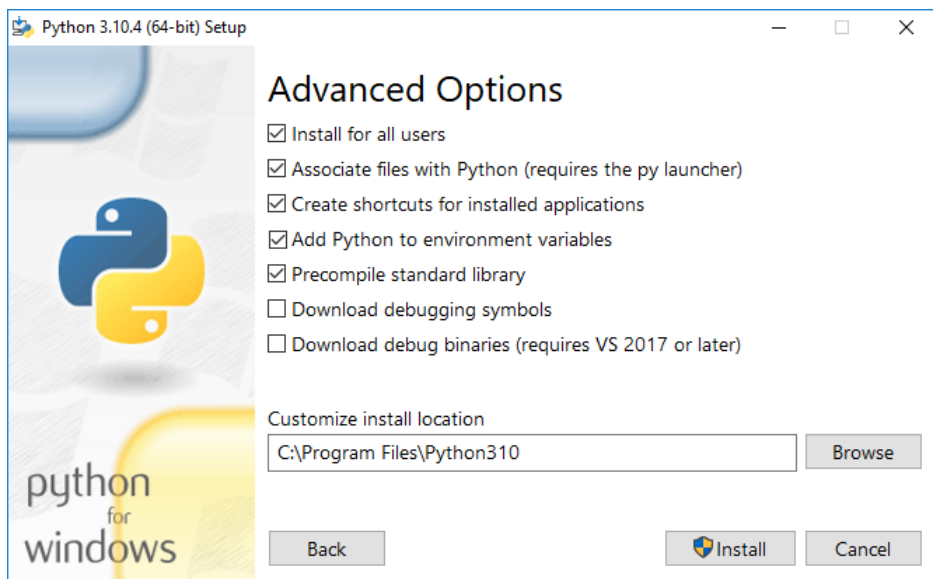
Pour télécharger l'installateur Python pour Windows, accédez à l'adresse : <https://www.python.org/downloads/> et cliquez ensuite sur le bouton **download** :



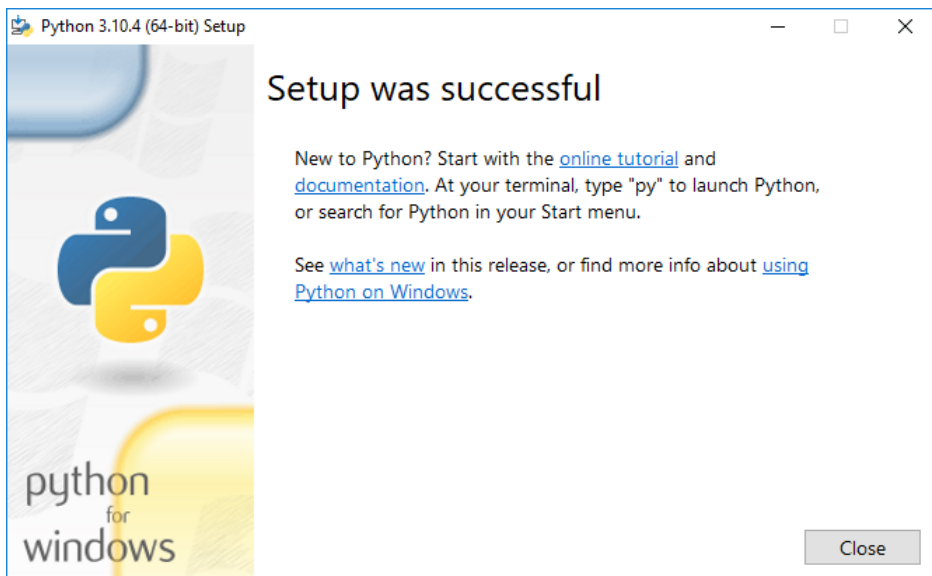
Lancer ensuite l'exécutable et cliquez sur **Customize installation** pour choisir une installation personnalisée :



Cochez ensuite les options : **add Python to environnement variables, install for all users...** comme le montre la figure suivante :



L'installation prendra à peu près une minute et vous verrez apparaître la fenêtre qui indique la **fin de l'installation** :



1.1.2 Installation de l'IDE Wing

En accédant à l'adresse : <https://wingware.com/downloads>, vous trouvez **trois version** de l'**IDE Wing**, choisissez alors la version **personal** qui est gratuite :

Wing Personal

A free Python IDE for students and hobbyists

- Simplified Debugger
- Limited Editor
- Some Code Inspection and Navigation
- Basic Project Management

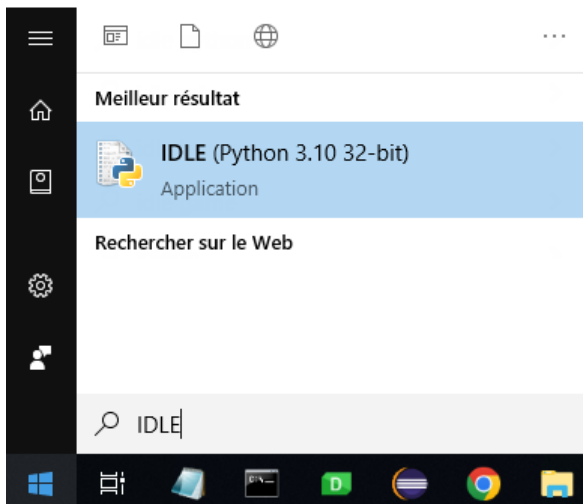
GET PERSONAL

Une fois l'exécutable téléchargé, lancez le et suivez les étapes d'installation automatique par défaut.

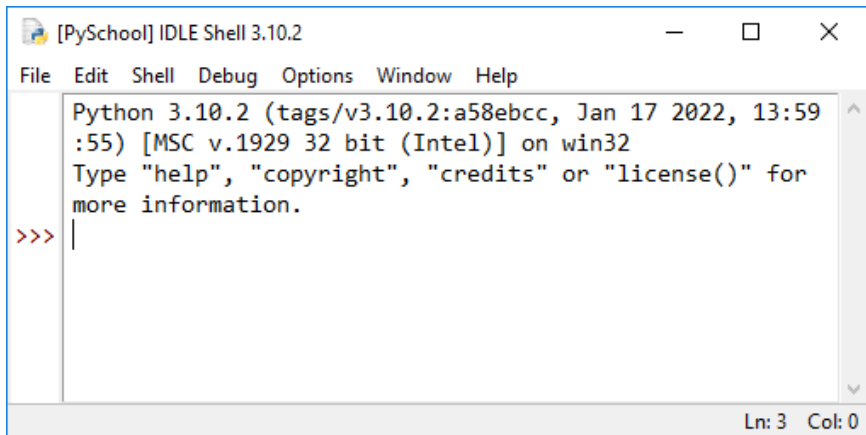
1.2 Premier programme en Python

1.2.1 Premier programme Python à l'aide de l'IDE intégré IDLE

Python est doté par défaut d'un IDE nommé **IDLE**, pour le lancer, il suffit de taper **IDLE** sur la zone recherche du menu démarrer :



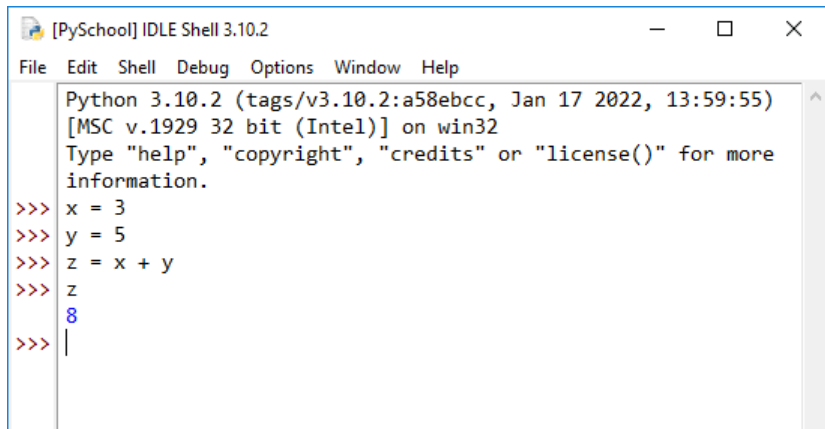
Une fois lancé, vous obtenez la vue suivante :



```
[PySchool] IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 13:59:55) [MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

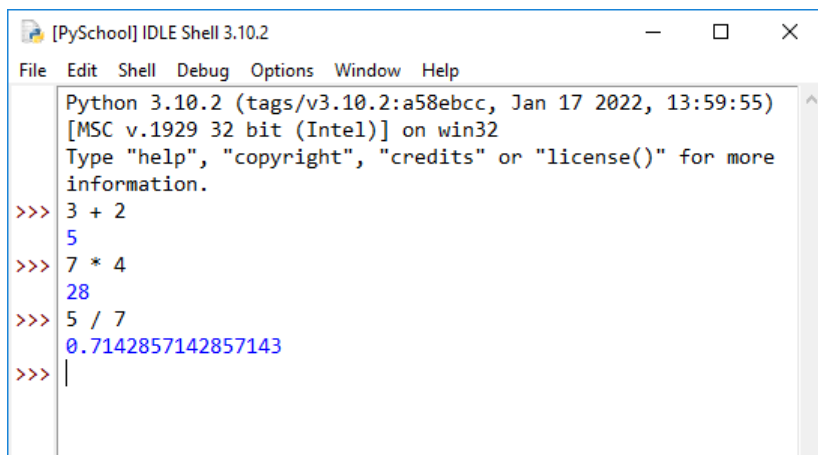
Comme vous le constatez, il s'agit d'un IDE très minimaliste. Amusons nous maintenant à faire quelque essais :

- A titre d'exemple on va définir **deux variables** **x** et **y** du type **entier** et on affiche leur somme **z = x + y** :



```
[PySchool] IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 13:59:55) [MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x = 3
>>> y = 5
>>> z = x + y
>>> z
8
>>> |
```

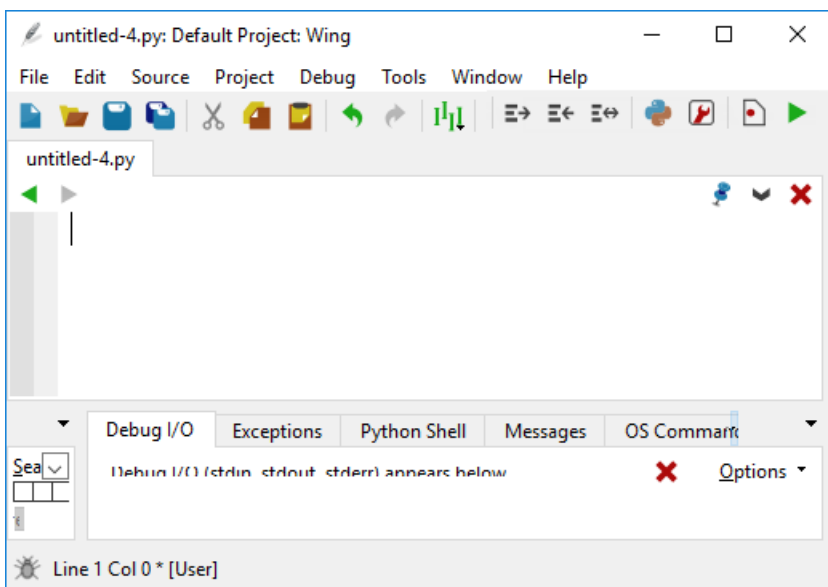
- L'outil IDLE peut aussi jouer le rôle d'une calculatrice :



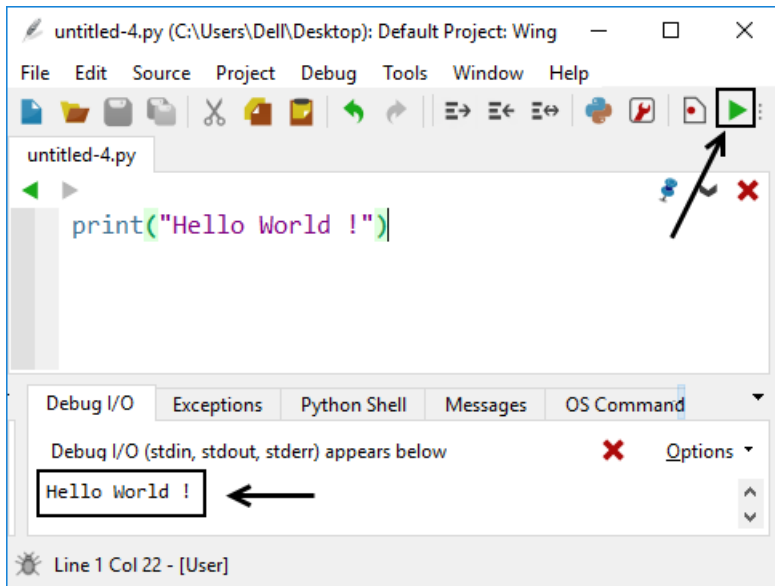
```
[PySchool] IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 13:59:55)
[MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> 3 + 2
5
>>> 7 * 4
28
>>> 5 / 7
0.7142857142857143
>>> |
```

1.2.2 Premier programme Python à l'aide de l'IDE wing

Si vous avez déjà installé l'IDE **Wing**, lancer le depuis le **menu démarré** et cliquez sur le menu **File->New** pour créer un **nouveau** projet :



Nous allons essayer de faire un petit script qui affiche le message « **Hello World !** ». A cet effet, il suffit d'utiliser la fonction **print** et cliquez sur l'icône sous forme de **petit triangle vert** :



Après avoir cliqué sur l'icône d'exécution du programme (petit **triangle vert**), vous verrez apparaître le message « **Hello World !** » en bas sur la **console**.

1.3 Les variables, commentaires & opérateurs en Python

1.3.1 les variables en Python

Contrairement à d'autres langages de programmation, Python n'a pas de commande pour déclarer une variable. Une variable est créée au moment où vous lui affectez une valeur.

Exemple. de variables en python

```
1 x = 7
2 y = "Albert"
3 print(x)
4 print(y)
```

Une variable python possède toujours un type, même s'il est non déclarée. le type se définit au moment de l'introduction de la variable et peut être changé par la suite, ce qui justifie le dynamisme et la puissance du langage Python

Exemple. Type d'une variable.

```
1 x = 3
2 # x est du type int
3 x = "Hello" # x est maintenant transformé en type string
```

1.3.2 Affichage d'une Variable

L'instruction `print` Python (on verra qu'il s'agit d'une fonction) est souvent utilisée pour générer la sortie des variables.

Exemple. affichage variable

```
1 x = 5
2 print(x) # affiche 5
```

On peut aussi ajouter un texte explicatif :

Exemple. affichage avec un texte explicatif

```
1 x = 5
2 print("La valeur de x est : ",x)
3 # affiche : La valeur de x est 5
```

1.3.3 Les commentaires en Python

1.3.3.1 Qu'est-ce qu'un commentaire en Python ?

Les langages de programmation fournissent une méthode pour l'insertion de commentaires au sein du code afin de fournir des informations supplémentaire. Un commentaire n'est autre qu'un texte qui sera ignoré lors de l'exécution du programme. Les commentaires peuvent être utilisés pour expliquer une partie compliquée d'un programme, ou pour mettre des indications dans le code, comme le code source, la version du langage ou script

1.3.3.2 Commentaire sur une seule ligne

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

Syntaxe

```
1 # Ceci est un commentaire qui sera ignoré à l'exécution
```

Exemple. de commentaire en python

```
1 # définir une variable de type entier
2 n = 5
3 # Affichage de la variable
4 print ("La valeur de n est :", n)
```

1.3.3.3 Commentaire sur plusieurs lignes

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

1.3.3.4 Syntaxe

```
1 """
2 Ceci est un commentaire
3 en plusieurs lignes
4 qui sera ignoré lors de l'exécution
5 """
```

Exemple. commentaire sur plusieurs lignes

```
1 """
2 Code source : tresfacile.net
3 date : septembre 2019
4 Auteur : Younes Derfoufi
5 """
6 [mon code python ici]
```

1.3.4 Les opérateurs en Python

1.3.4.1 Les différents types d'opérateurs en Python

Les opérateurs sont utilisés en Python pour effectuer des opérations sur les variables et les valeurs associées. Python classe les opérateurs selon les groupes suivants :

1. Opérateurs arithmétiques
2. Opérateurs d'assignation
3. Opérateurs de comparaison
4. Opérateurs logiques

1.3.4.2 Les opérateurs arithmétiques

Les opérateurs arithmétiques sont utilisés en Python pour effectuer des opérations de calcul sur les variables comme addition, multiplication, division

Opérateur	Description
'+'	addition
'_'	soustraction
'*'	multiplication
'/'	division
'%'	modulo (reste de la division euclidienne)
'**'	Exponentiation
'//'	quotient de la division euclidienne

1.3.4.3 Les opérateurs d'assignation

Les opérateurs d'assignation sont utilisés en Python pour assigner des valeurs aux variables :

Opérateurs Exemple Explication

Opérateur	Exemple	Explication
=	x = 7	x prends la valeur 7
+ =	x + = 5	x = x + 5
- =	x - = 5	x = x -5
* =	x * = 5	x = x *5
/ =	x / = 5	x = x / 5
% =	x % = 5	reste de la division euclidienne de x par 5
// =	x // = 5	quotient de la division euclidienne de x par 5
** =	x ** = 3	x = x **3 (x^3 ie $x*x*x$)
& =	x & = 5	x = x &5 (& désigne l'opérateur binaire)

1.3.4.4 Opérateurs de comparaison

Les opérateurs de comparaison sont utilisé en Python pour comparer les variables :

Opérateur	Description
<code>= =</code>	opérateur d'égalité
<code>! =</code>	opérateur différent
<code>></code>	opérateur supérieur
<code><</code>	opérateur inférieur
<code>> =</code>	opérateur supérieur ou égale
<code>< =</code>	opérateur inférieur ou égale

1.3.4.5 Opérateurs logiques

Opérateur	Description
<code>and</code>	et logique
<code>or</code>	ou logique
<code>not</code>	Négation logique

1.4 Les fonctions en Python

Le langage Python possède déjà des fonctions prédéfinies comme `print()` pour afficher du texte ou une variable, `input()` pour lire une saisie clavier... Mais il offre à l'utilisateur la possibilité de créer ses propres fonctions :

Exemple. fonction qui renvoie le double d'un nombre

```

1 def maFonction(x) :
2     return 2*x
3 print("Le double de 5 est : " , maFonction(5))
4 # affiche : Le double de 5 est : 10

```

Remarque importante à propos de la syntaxe !

```

def maFonction(x):
    → return 2*x
    print("Le double de 5 est : " , maFonction(5))

```

Remarquez bien le décalage ici qui montre que l'instruction **return** est située à l'intérieur de la fonction. Faute de quoi on reçoit un message d'erreur

1.5 Structures de contrôles

1.5.1 La structure sélective If ... Else ...

La structure sélective **if ...else**, permet d'exécuter un ensemble d'instructions lorsqu'une condition est réalisée.

Syntaxe :

```
1 if(condition) :
2     instructions ...
3 else :
4     autres instructions ...
```

Exemple. structure if ... else...

```
1 # coding : utf-8
2 age = 19
3 if(age >= 18) :
4     print("Vous êtes majeur !")
5 else :
6     print("Vous êtes mineur !")
7 # affiche vous êtes majeur
```

1.5.2 L'instruction elif

L'instruction **elif** est employée généralement lorsque l'exception comporte **2 ou plusieurs cas à distinguer**. Dans notre exemple ci-dessus l'exception est **age < 18** qui correspond au cas mineur. Or le cas mineur comporte les deux cas :

1. **Enfance** $\text{age} < 14$
2. **Adolescence** $14 < \text{age} < 18$

L'instruction **else** sélectionne la condition contraire qui est **age < 18** et donc ne peut distinguer entre les deux cas **enfance** et **adolescence**. Ainsi pour palier à ce problème, on utilise l'instruction **elif** :

Exemple. instruction elif

```
1 -*- coding : utf-8 -*-
2 age = int(input('tapez votre age : '))
3 if(age >= 18) :
4     print("Vous êtes majeur !")
5 elif(age < 15) :
6     print("Vous êtes trop petit !")
7 else :
8     print("Vous êtes adolescent !")
```


1.5.3 La structure répétitive For ...

La boucle for, permet d'exécuter des instructions répétées. Sa syntaxe est :

```
1 # -*- coding : utf-8 -*-
2 for compteur in range(début_compteur, fin_compteur) :
3     instructions ...
```

Exemple. affichage des 10 premiers nombres

```
1 # -*- coding : utf-8 -*-
2 for i in range(1,11) :
3     print(i)
4 #affiche les 10 premiers nombres 1 , 2 , ... , 10
```

Remarque 1. Noter que dans la boucle **for i in range(1,n)** le dernier qui est **n** n'est pas inclus! Cela veut dire que la boucle s'arrête à l'ordre **n-1**.

1.5.4 La structure répétitive While

La structure **while** permet d'exécuter un ensemble d'instructions tant qu'une condition est réalisée et que l'exécution s'arrête lorsque la condition n'est plus satisfaite. Sa syntaxe est :

```
1 while ( condition ) :
2     intructions ...
```

Exemple. affichage des 10 premiers entiers avec la boucle while

```
1 i = 1
2 while (i <= 10) :
3     print(i)
4     i = i + 1
```

1.5.5 Les exceptions en Python (Try Except)

1.5.5.1 Exemple introductif

Considérons le code suivant qui permet de calculer le quotient de deux nombres a et b :

```
1 a = int(input("Tapez la valeur de a : "))
2 b = int(input("Tapez la valeur de b : "))
3 print("Le quotient de a par b est : a/b = " , a/b)
```

Si vous exécutez le code ci-dessus en donnant **a = 6** et **b = 3**, le programme renvoie :

Le quotient de a par b est : a/b = 2.0

Aucun problème ! Mais si l'utilisateur donne **a = 6** et **b = 0** le programme renvoie le message d'erreur :

builtins.ZeroDivisionError : division by zero.

En plus l'interpréteur python arrête l'exécution du code

Afin d'éviter ce message d'erreur, et continuer à exécuter la suite du code, on utilise la structure **Try ... except**

1. **Le bloc try** permet de tester un bloc de code s'il contient des erreurs ou non et ne l'exécute que s'il ne contient aucune erreur ! Dans le cas contraire le programme ignore la totalité du code dans ce bloc et passe au bloc suivant **except**.
2. **Le bloc except**, vous permet de gérer l'erreur.
3. **Le bloc finally**, vous permet d'exécuter du code, quel que soit le résultat des blocs **try** et **except**.

1.5.5.2 Gestion des exceptions

Lorsqu'une erreur se produit, ou exception comme nous l'appelons, Python s'arrête normalement et génère un message d'erreur.

Ces exceptions peuvent être gérées à l'aide de l'instruction **try** :

Exemple. .

```
1 a = int(input("Tapez la valeur de a : "))
2 b = int(input("Tapez la valeur de b : "))
3 try :
4     print("Le quotient de a par b est : a/b = " , a/b)
5 except :
6     print("Veuillez choisir une valeur b non nulle !")
```

Dans ce cas si vous donnez **a = 6** et **b = 0**, le programme **ignore** le code du **bloc try** après avoir détecté une **erreur** et passe automatiquement au code du **bloc except** et renvoie donc :

Veuillez choisir une valeur b non nulle !

1.5.5.3 Exception via une instruction raise

On peut se demander maintenant s'il est possible de lever une exception sans rencontrer une erreur. Exemple pour un programme qui demande à l'utilisateur de taper son âge et de lever une exception si l'âge est < 18 ans ! Bien entendu un âge tapé qui est inférieur à 18 ans est une opération

qui ne contient aucune erreur, et pourtant on peut quand même en lever une :

```
1 try :
2     age = int(input("Veuillez saisir votre age"))
3     if age < 18 :
4         raise ValueError
5     else :
6         print("age valide")
7 except :
8     print("age non valide !")
```

1.5.6 L’instruction finally

L’instruction **finally** est utilisée pour exécuter des instructions quelque soit les erreurs générées ou non. Dans tous les cas (présence d’erreurs ou non !) l’instruction déclarée dans le **block finally** sera **exécutée**.

Syntaxe :

```
1 try :
2     # bloc de code pouvant probablement lever une exception
3 finally :
4     # bloc de code qui sera toujours exécuté
```

1.6 Les chaînes de caractères en Python

1.6.1 Définir une chaîne de caractère en Python

Comme tous les autres langages de programmations, les chaînes de caractères en python sont des variables sous forme d’expressions entourées de guillemets simples ou de guillemets doubles. "CRMEF Oujda" est identique à 'CRMEF Oujda'.

Les chaînes de caractères peuvent être affichées à l’écran en utilisant la fonction d’impression :

```
1 print(nom_de_la_variable_chaine)
```

Comme beaucoup d’autres langages de programmations populaires, les **chaînes de caractères** en Python sont des **tableaux d’octets** représentant des caractères Unicode. Cependant, Python ne possède pas de type de données caractère (**char**) comme char type en C, un seul caractère est simplement une chaîne de longueur 1. Les crochets peuvent être utilisés pour accéder aux éléments de la chaîne.

1.6.2 Longueur d'une chaîne de caractères

La longueur d'une chaîne de caractère est par définition le nombre de caractères qui composent la chaîne. Pour obtenir la longueur d'une chaîne de caractère, on utilise la **méthode** `len()`

Exemple. (longueur de la chaîne `s = "CRMEF OUJDA"`)

```
1 s = "CRMEF OUJDA"
2 print("La longueur de s est :", len(s)) # affiche La
   longueur de s est : 11
```

1.6.3 Accéder aux éléments d'une chaîne de caractères

Pour accéder à un élément d'une chaîne de **caractère** `s`, on utilise la syntaxe :

```
1 s[index_du_caractère]
```

Exemple. Obtenir le premier et deuxième caractère de la chaîne (rappelez-vous que le premier caractère se trouve à la position 0) :

```
1 s = "CRMEF OUJDA"
2 print("premier caractère de s est :", s[0])# affiche : "
   premier caractère de s est C
3 print("deuxième caractère de s est :", s[1])# affiche : "
   deuxième caractère de s est R
```

Exemple. (affichage total des caractères d'une chaîne à l'aide de la **méthode** `len()`)

```
1 s = "Python"
2 for i in range(0 , len(s)) :
3     print(s[i])
4 # affiche :
5 """
6 P
7 y
8 t
9 h
10 o
11 n
12 """
```

Exemple. (affichage total des caractères d'une chaîne via la méthode d'itérateur)

```
1 s = "Python"
2 for x in s :
3     print(x)
4 # affiche :
5 " "
6 P
7 y
8 t
9 h
10 o
11 n
12 " " "
```

1.6.4 Opération sur les chaînes de caractères

1.6.4.1 Concaténation de deux chaînes de caractères

Pour faire la **concaténation** de deux **chaînes** de **cractères**, on utilise l'**opérateur** '+' :

Exemple.

```
1 s1 = "Learn "
2 s2 = "Python"
3 # concaténation de s1 et s2
4 s = s1 + s2
5 print(s) # affiche : 'Learn Python' "
```

1.6.5 Extraire une sous chaîne de caractères

On **extraît** une **sous chaîne** de **s** depuis la $i^{ème}$ position jusqu'à la $j^{ème}$ non incluse en utilisant la syntaxe :

```
1 substring = string[i : j] "
```

Exemple.

```
1 s = "Python"
2 substring = s[2 : 5]
3 print(substring) # affiche : 'tho' "
```

1.6.6 Les fonctions de chaînes de caractères en Python

Le langage Python est doté d'un grand nombre de fonctions permettant la manipulation des chaînes de caractères : calcul de la **longueur de la**

chaîne, transformation en **majuscule** et **minuscule**, extraire une **sosus** chaîne...En voici une liste non exhaustive :

1. **capitalize()** : Met en majuscule la première lettre de la chaîne
2. **center(largeur, remplissage)** : Retourne une chaîne complétée par des espaces avec la chaîne d'origine centrée sur le total des colonnes de largeur.
3. **count(str, beg = 0, end = len (chaîne))** : Compte le nombre de fois où str se produit dans une chaîne ou dans une sous-chaîne si le début de l'index de début et la fin de l'index de fin sont indiqués.
4. **decode(encodage = 'UTF-8', erreurs = 'strict')** : Décode la chaîne en utilisant le codec enregistré pour le codage. Le codage par défaut correspond au codage de chaîne par défaut.
5. **encode(encoding = 'UTF-8', errors = 'strict')** : Retourne la version encodée de la chaîne ; en cas d'erreur, la valeur par défaut est de générer une valeur ValueError sauf si des erreurs sont indiquées avec "ignore" ou "remplace".
6. **endswith(suffixe, début = 0, fin = len(chaîne))** : Détermine si une chaîne ou une sous-chaîne de chaîne (si les index de début et de fin d'index de fin sont indiqués) se termine par un suffixe ; renvoie vrai si oui et faux sinon.
7. **expandtabs(tabsize = 8)** : Développe les onglets d'une chaîne en plusieurs espaces ; La valeur par défaut est 8 espaces par onglet si tabsize n'est pas fourni.
8. **find(str, beg = 0 end = len (chaîne))** : Déterminer si str apparaît dans une chaîne ou dans une sous-chaîne de chaînes si l'index de début et l'index de fin sont spécifiés, end renvoie return s'il est trouvé et -1 dans le cas contraire.
9. **format(string s)** : remplace les accolades par la variable string s (voir exemple ci-dessous : 1.6.6)
10. **index(str, beg = 0, end = len (chaîne))** : Identique à find (), mais déclenche une exception si str n'est pas trouvé.
11. **isalnum()** : Retourne true si la chaîne a au moins 1 caractère et que tous les caractères sont alphanumériques et false sinon.
12. **isalpha()** : Retourne vrai si la chaîne a au moins 1 caractère et que tous les caractères sont alphabétiques et faux sinon.
13. **isdigit()** : Renvoie true si la chaîne ne contient que des chiffres et false sinon.

14. **islower()** : Retourne true si la chaîne a au moins 1 caractère en casse et que tous les caractères en casse sont en minuscule et false sinon.
15. **isnumeric()** : Renvoie true si une chaîne unicode contient uniquement des caractères numériques et false sinon.
16. **isspace()** : Renvoie true si la chaîne ne contient que des caractères d'espacement et false sinon.
17. **istitle()** : Retourne true si la chaîne est correctement "titlecased" et false sinon.
18. **isupper()** : Renvoie true si string contient au moins un caractère et que tous les caractères sont en majuscule et false sinon.
19. **join(seq)** : Fusionne (concatène) les représentations sous forme de chaîne d'éléments en séquence seq dans une chaîne, avec chaîne de séparation.
20. **len(chaîne)** : Retourne la longueur de la chaîne
21. **ljust(largeur [, remplissage])** : Renvoie une chaîne complétée par des espaces avec la chaîne d'origine justifiée à gauche pour un total de colonnes de largeur.
22. **lower()** : Convertit toutes les lettres majuscules d'une chaîne en minuscules.
23. **lstrip()** : Supprime tous les espaces en début de chaîne.
24. **maketrans()** : Renvoie une table de traduction à utiliser dans la fonction de traduction.
25. **max(str)** : Renvoie le caractère alphabétique maximal de la chaîne str.
26. **min(str)** : Renvoie le caractère alphabétique minimal de la chaîne str.
27. **replace(ancien, nouveau [, max])** : Remplace toutes les occurrences de old dans string par new ou au maximum max si max donné.
28. **rfind(str, beg = 0, end = len(chaîne))** : Identique à find(), mais recherche en arrière dans string.
29. **rindex(str, beg = 0, end = len(chaîne))** : Identique à index(), mais recherche en arrière dans string.
30. **rjust(largeur, [, remplissage])** : Renvoie une chaîne complétée par des espaces avec la chaîne d'origine justifiée à droite, avec un total de colonnes de largeur.

31. **rstrip()** : Supprime tous les espaces de fin de chaîne.
32. **split(str = "", num = string.count (str))** : Divise la chaîne en fonction du délimiteur str (espace si non fourni) et renvoie la liste des sous-chaînes; divisé en sous-chaînes au maximum, le cas échéant.
33. **splitlines(num = string.count ('\ n'))** : Fractionne la chaîne de tous les NEWLINE (ou num) et renvoie une liste de chaque ligne sans les NEWLINE.
34. **startswith(str, beg = 0, end = len (chaîne))** : Détermine si string ou une sous-chaîne de chaîne (si les index de début et de fin d'index de fin sont indiqués) commence par la sous-chaîne str; renvoie vrai si oui et faux sinon.
35. **strip([chars])** : Effectue **lstrip ()** et **rstrip ()** sur chaîne.
36. **swapcase()** : Inverse la casse de toutes les lettres d'une chaîne.
37. **title()** : Retourne la version "titlecased" de la chaîne, c'est-à-dire que tous les mots commencent par une majuscule et le reste est en minuscule.
38. **translate(table, deletechars = "")** : Traduit la chaîne en fonction de la table de traduction str (256 caractères), en supprimant celles de la chaîne del.
39. **upper()** : Convertit les lettres minuscules d'une chaîne en majuscules.
40. **zfill(largeur)** : Renvoie la chaîne d'origine laissée avec des zéros à un total de caractères de largeur; destiné aux nombres, **zfill ()** conserve tout signe donné (moins un zéro).
41. **isdecimal()** : Renvoie true si une chaîne unicode ne contient que des caractères décimaux et false sinon.
42. **s[i : j]** : Extrait la sous chaîne de s depuis la ième position jusqu'à la jème non incluse
43. **s[i :]** : Extrait la sous chaîne de s depuis la ième position jusqu'à la fin de la chaîne
44. **s[: j]** : Extrait la sous chaîne de s depuis le début jusqu'à la jème position non incluse

Exemple. transformation d'une chaîne en minuscule

```

1 s="CRMEF OUJDA"
2 s = s.lower()
3 print(s) # affiche crmef oujda

```


Exemple. remplacement d'une occurrence par une autre

```
1 s="CRMEF OUJDA"
2 s = s.replace("CRMEF", "ENS")
3 print(s) # affiche ENS OUJDA
```

Exemple. Nombre de caractères d'une chaîne

```
1 #-*- coding : utf-8 -*-
2 s = "CRMEF OUJDA"
3 n = len(s)
4 print("le nombre de caractères de la chaîne s est : " , n)
5 # affiche le nombre de caractères de la chaîne s est : 11
```

Exemple. String.format

```
1 nom = "David"
2 age = 37
3 s = 'Bonjour , {}, vous avez {} ans'.format(nom, age)
4 print(s) # affiche 'Bonjour , David, vous avez 37 ans'
```

Exemple. extraire une sous chaîne

```
1 s = "CRMEF OUJDA"
2 s1 = s[6:9]
3 print(s1) # affiche OUJ
4 s2 = s[6:]
5 print(s2) # affiche OUJDA
6 s3 = s[:4]
7 print(s3) # affiche CRME
```

1.7 Les listes en Python

1.7.1 Création d'une liste en Python

Une liste en Python est un type de données ordonnée et modifiable qui fait partie des collections . En Python, les listes sont écrites entre crochets.

Exemple. —

```
1 #Création d'une liste
2 myList = ["Python", "Java", "PHP"]
3 # Affichage de la liste
4 print (myList)
```

1.7.2 Accès aux éléments d'une liste.

Vous accédez aux éléments d'une liste Python, en vous référant au numéro d'index :

Exemple. Imprimer le 3ème élément de la liste :

```
1 myList = ["Python", "Java", "PHP"]
2 print(myList[2]) # Affiche 'PHP'
```

1.7.3 Changer la valeur d'un élément de la liste

Pour modifier la valeur d'un élément spécifique, reportez-vous au numéro d'index :

Exemple. Changer le deuxième élément :

```
1 myList = ["Python", "Java", "PHP"]
2 myList[1] = "Oracle"
3 print(myList) # affiche : ['Python', 'Oracle', 'PHP']
```

1.7.4 Parcourir les éléments d'une liste Python

Vous pouvez parcourir les éléments d'une liste en Python, en utilisant une boucle for :

Exemple. Imprimer tous les éléments de la liste, un par un :

```
1 myList = ["Formation", "Python", "au CRMEF OUJDA"]
2 for x in myList :
3     # Afficher tous les éléments de la liste un par un
4     print(x)
```

1.7.5 Longueur d'une liste Python

Pour déterminer le nombre d'éléments d'une liste, utilisez la méthode len() :

Exemple. Imprimer le nombre d'éléments de la liste :

```
1 myList = ["Python", "Java", "PHP"]
2 print("La longueur de ma liste est", len(myList))
3 # Affiche : La longueur de ma liste est 3
```

1.7.6 Ajouter ou supprimer des éléments à la liste

1.7.6.1 Ajouter un un élément à une liste Python

– Pour ajouter un élément à la fin de la liste, on utilise la méthode `append()` :

Exemple. ajouter un élément à la liste avec la méthode `append()` :

```
1 myList = ["Formation", "Python", "au CRMEF"]
2 myList.append("OUJDA")
3 print (myList)
4 #Affiche : ["Formation", "Python", "au CRMEF", "OUJDA"]
```

– Pour ajouter un élément à l'index spécifié, utilisez la méthode `insert()` :

Exemple. Insérer un élément en deuxième position :

```
1 myList = ["Python", "Java", "PHP"]
2 myList.insert(1, "C++")
3 print (myList)
4 # Affiche : ["Python", "C++", "Java", "PHP"]
```

1.7.6.2 Retirer un élément d'une liste Python

Il existe plusieurs méthodes pour supprimer des éléments d'une liste :

1. La méthode `remove()` supprime un élément spécifié.
2. La méthode `pop()` supprime un élément en spécifiant son index (ou le dernier élément si aucun index n'est spécifié)
3. Le mot clé `del` supprime l'élément à l'index spécifié(`del` permet également de supprimer complètement la liste)
4. La méthode `clear()` vide la liste :

Exemple. suppression d'un élément spécifié avec la méthode `remove()`

```
1 myList = ["Python", "Java", "PHP"]
2 myList.remove("Java")
3 print (myList) # Affiche : ["Python", "PHP"]
```

Exemple. Suppression d'un élément d'index spécifié avec la méthode `pop()`

```
1 myList = ["Python", "Java", "PHP"]
2 myList.pop(0)
3 print (myList) # Affiche : ["Java", "PHP"]
```

Exemple. suppression d'élément à un index spécifié avec la méthode **del** :

```
1 myList = ["Python", "Java", "PHP"]
2 del myList[1]
3 print (myList) # affiche : ["Python", "PHP"]
```

Remarque 2. Le mot clé **del** peut également **supprimer** complètement la liste :

Exemple. suppression d'une liste

```
1 myList = ["Python", "Java", "PHP"]
2 del myList
3 print (myList) # cela causera une erreur car "myList" n'
                 existe plus.
```

Exemple. vider une liste

```
1 myList = ["Python", "Java", "PHP"]
2 myList.clear ()
3 print (myList) # cela affiche des crochets vides [ ] car "
                 myList" est vide.
```

1.7.6.3 Transformer une chaîne de caractères en une liste

Le langage Python est doté de la méthode **split()** qui permet de transformer une chaîne de caractères en une liste :

Exemple. methode **split()**

```
1 s = "CRMEF OUJDA est un centre de formation des
    enseignants"
2 liste = s.split()
3 print(liste)
4 # Affiche :['CRMEF', 'OUJDA', 'est ', 'un', 'centre', 'de',
             'formation', 'des', 'enseignants']
```

Exemple. méthode **split()** avec caractère de séparation

```
1 s = "Bonjour, les examens auront lieu le 15/07/19, au
    CRMEF OUJDA"
2 liste = s.split(',')
3 print(liste)
4 # Affiche :['Bonjour', ' les examens auront lieu le
             15/07/19', ' au CRMEF OUJDA']
```

1.7.6.4 Transformer une liste en une chaîne de caractères

Nous allons maintenant comment réaliser l'opération inverse : transformer une liste en une chaîne de caractères ! La méthode qui peut jouer l'affaire dans ce cas c'est la méthode **join()** :

Exemple. la méthode **join()**

```
1 l = ['Bonjour', ' les examents auront lieu le 15/07/19', '
      au CRMEF OUJDA']
2 s = ":".join(l)
3 print(s)
4 # Affiche : Bonjour : les examents auront lieu le 15/07/19 :
      au CRMEF OUJDA
5 # On peut aussi omettre le motif de séparation et mettre
      simplement un espace
6 s = " ".join(l)
7 print(s)
8 # Affiche : Bonjour  les examents auront lieu le 15/07/19
      au CRMEF OUJDA
```

1.7.6.5 Compter la fréquence des éléments d'une liste

On souhaite parfois obtenir des informations concernant le nombre de répétition des termes d'une liste. Pour ce faire on doit importer le **module** **collection** et utiliser la méthode **Counter()** :

Exemple. la méthode **Counter()**

```
1 from collections import Counter
2 l = ['crmef', 'oujda', 'rabat', 'oujda']
3 print(Counter(l))
4 # Affiche : Counter({'oujda': 2, 'crmef': 1, 'rabat': 1})
```

1.7.7 Les différentes méthodes destinées aux listes Python

Python a un ensemble de méthodes intégrées que vous pouvez utiliser pour manipuler listes d'une façon très souple :

1. **append()** : ajoute un élément à la fin de la liste
2. **clear()** : supprime tous les éléments de la liste
3. **copy()** : retourne une copie de la liste
4. **count()** : retourne le nombre d'éléments avec la valeur spécifiée
5. **extend()** : ajoute les éléments d'une liste (ou de tout élément itérable) à la fin de la liste actuelle

6. `index()` : retourne l'index du premier élément avec la valeur spécifiée.
7. `insert()` : ajoute un élément à la position spécifiée
8. `pop()` : supprime l'élément à la position spécifiée
9. `remove()` : supprime l'élément avec la valeur spécifiée
10. `reverse()` : inverse l'ordre de la liste
11. `sort()` : trie la liste

1.8 Les tuples

1.8.1 Définir un tuple en Python

Un tuple est une collection ordonnée et non modifiable (n-uplets en mathématiques). En Python, les tuples sont écrits avec des parenthèses.

Exemple. Création d'un tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 print(myTuple)
3 # Affiche : ('cartable', 'cahier', 'livre')
```

1.8.2 Accéder aux élément d'un tuple

Vous pouvez accéder aux éléments d'un tuple en vous référant au numéro d'index, entre crochets :

Exemple. Accéder à l'élément qui se trouve en position 1 :

```
1 myTuple = ("cartable", "cahier", "livre")
2 print(myTuple[1])
3 # Affiche : cahier
```

Remarque 3. Une fois un tuple est créé, vous ne pouvez pas **modifier ses valeurs**. Les tuples **sont immuables**.

1.8.3 Boucle à travers un tuple

Vous pouvez parcourir les éléments d'un tuple en utilisant une boucle `for`.

Exemple. Parcourez les éléments et imprimez les valeurs :

```
1 myTuple = ("cartable", "cahier", "livre")
2 for x in myTuple :
3     print (x)
4 # Affiche tous les éléments du tuple.
```

1.8.4 Vérifier si un élément existe dans un tuple

Pour déterminer si un élément spécifié est présent dans un tuple, utilisez le mot-clé `in` :

Exemple. Vérifiez si `"cartable"` est présent dans le tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 if ("cartable" in myTuple) :
3     print("Oui, 'cartable' est dans myTuple")
```

1.8.5 Longueur d'un tuple

La longueur d'un tuple désigne le nombre d'éléments qui le compose. Pour déterminer la longueur d'un tuple en Python, on utilise la méthode `len()` :

Exemple. nombre d'éléments d'un tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 print(len(myTuple))
3 # Affiche 3
```

1.8.6 Ajout ou suppression d'éléments impossible à un tuple

Remarque 4. Une fois qu'un tuple est créé, on ne peut lui ajouter d'éléments. Les tuples sont immuables.

Exemple. Ajout d'éléments **impossible** à un tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 myTuple [3] = "Stylo" # Ceci provoquera une erreur !
```

1.8.7 Suppression d'un tuple

Les tuples ne sont pas modifiables, vous ne pouvez donc pas en supprimer d'éléments, mais vous pouvez supprimer complètement le tuple à l'aide du mot clé **del** :

Exemple. Supprimer complètement un tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 del myTuple
3 print(myTuple) #cela générera une erreur car le tuple n'
    existe plus
```

1.8.8 Création d'un tuple en utilisant le constructeur tuple()

Il existe une autre méthode pour créer un tuple qui consiste à utiliser le constructeur **tuple()**.

Exemple. Création d'un tuple en utilisant le constructeur **tuple()** :

```
1 myTuple = tuple(("cartable", "cahier", "livre"))
2 # notez les doubles parenthèses rondes
3 print(myTuple)
```

1.8.9 Méthodes associées à un tuple

Python a deux méthodes intégrées que vous pouvez utiliser sur des n-uplets.

1. **count ()** : retourne le nombre de fois qu'une valeur spécifiée apparaît dans un tuple.
2. **index ()** : recherche dans le tuple une valeur spécifiée et renvoie la position de l'endroit où il a été trouvé.

1.9 Les tableaux (array)

1.9.1 A propos du module array

Le module **array** n'est qu'un **wrapper** emprunté du langage C. La structure d'un objet **array** est proche à celle d'un objet liste. Fondamentalement, les listes Python sont très flexibles et peuvent contenir des données arbitraires **complètement hétérogènes**, comme par exemple **L = [11, 'a', 'crmef', 'oujda']**... et elles peuvent être ajoutées de manière très

efficace en toute simplicité. Si vous devez réduire et agrandir votre liste de manière efficace et sans tracas, c'est le type d'objet que vous devez choisir. Mais ils utilisent beaucoup plus d'espace que les tableaux C.

Le type `array.array`, d'autre part, n'est qu'un mince wrapper des tableaux en C. Il ne peut contenir que des données homogènes, toutes du même type, et il n'utilise donc que `sizeof (objet) * longueur octets` de mémoire. Généralement, vous devez l'utiliser lorsque vous devez exposer un tableau C à une extension ou à un appel système (par exemple, `ioctl` ou `fcntl`).

`array.array` est également un moyen raisonnable de représenter une chaîne modifiable en Python (`array('B', bytes)`). Cependant, Python 2.7+ et 3.x propose une chaîne d'octets mutable comme **`bytearray`**.

Pour faire court : `array.array` est utile lorsque vous avez besoin d'un tableau de données homogène pour des raisons autres que de faire des calculs.

1.9.2 Syntaxe & création d'un array

Pour créer un tableau (`array`), on utilise la méthode `array()` avec indication du type d'éléments :

```
1 array(type de données , liste de valeurs)
```

Exemple. tableau du type `int`

```
1 L = [23 , 11 , 7 , 19]
2 Tab = array('i', [23 , 11 , 7 , 19])
```

Avec :

- **'i'** : indique le type du tableau (`int`)
- **L** : liste des valeurs qui compose le tableau.

1.9.3 Type de données pour un tableau

Nous venons de voir comment créer un tableau du type `int` en déclarant l'indicateur du type par **'i'**. Cependant, pour déclarer un tableau, il n'y a pas que le type `int`. En voici un tableau qui résumant tous les types qui peuvent être utiliser pour déclarer un tableau :

Code	Type C	Type Python	Taille en octets
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Caractère Unicode	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

1.10 Les dictionnaires

1.10.1 Définir un dictionnaire en Python

Un dictionnaire est une implémentation par Python d'une structure de données semblable à un tableau associatif. Un dictionnaire consiste en une collection de paires clé-valeur. Chaque paire clé-valeur fait attacher la clé à sa valeur associée.

On peut définir un dictionnaire en entourant des accolades `{ }` une liste de **paires clé-valeur** séparées par des virgules.

Syntaxe :

```
1 dic = {key1 : valeur1 , key2 : valeur2 , key3 : valeur3 , ...}
```

Pour accéder à une valeur à partir du dictionnaire, on utilise le nom du dictionnaire suivi de la clé correspondante entre crochets :

```
1 dic = {key1 : valeur1 , key2 : valeur2 , key3 : valeur3 , ...}
2 print(dic[key1]) # affiche valeur1
```

Exemple. Annuaire téléphonique

```
1 phoneBook = {"Majid" : "0556683531" , "Tomas" : "0537773332" , "
                Bernard" : "0668793338" , "Hafid" : "066445566"}
2 print(phoneBook["Majid"]) # affiche 0556683531
```

1.10.2 Parcourir les valeurs et les clés d'un dictionnaire Python

Un dictionnaire en Python est doté d'une méthode nommée **values()** qui permet de parcourir ses valeurs, et d'une autre nommée **keys()** permettant de parcourir ses clés.

Exemple. parcourt des valeurs d'un dictionnaire

```
1 phoneBook={"Majid" : "0556633558", "Tomas" : "0587958414", "
    Bernard" : "0669584758"}
2 for valeur in phoneBook.values() :
3     print(valeur)
```

Exemple. parcourt des clés d'un dictionnaire

```
1 phoneBook={"Majid" : "0556633558", "Tomas" : "0587958414", "
    Bernard" : "0669584758"}
2 for key in phoneBook.keys() :
3     print(key)
```

Remarque 5. On peut aussi parcourir les clés et les valeurs en même temps en passant à la méthode **items()**

Exemple. parcourt des clés et des valeurs

```
1 phoneBook={"Majid" : "0556633558", "Tomas" : "0587958414", "
    Bernard" : "0669584758"}
2 for key , valeur in phoneBook.items() :
3     print(key , valeur)
```

1.10.3 Mettre à jour, ajouter ou supprimer des éléments d'un dictionnaire

1.10.3.1 Mettre à jour un élément du dictionnaire

On peut mettre à jour un élément du dictionnaire directement en affectant une valeur à une clé :

Exemple. gestionnaire d'un stock

```
1 stock={"Laptop" :15, "Imprimante" :35,"Tablette" :27}
2
3 #modification de la valeur associée à la clé "Imprimante"
4 stock["Imprimante"]=42
5 print(stock)
6 # affiche : {'Laptop': 15, 'Imprimante': 42, 'Tablette':
    27}
```

1.10.3.2 Ajouter un élément au dictionnaire

Dans le cas d'une clé inexistante, la même méthode citée ci-dessus, permet d'ajouter des éléments au dictionnaire :

Exemple. Ajouter un élément au stock

```
1 stock={"Laptop":15, "Imprimante":35,"Tablette":27}
2
3 # Ajout de l'élément "Ipad":18
4 stock["Ipad"]=18
5 print(stock)
6 # affiche : {'Laptop': 15, 'Imprimante': 35, 'Tablette':
    27, 'Ipad':18}
```

1.10.3.3 Supprimer un élément du dictionnaire

On peut supprimer un élément du dictionnaire en indiquant sa clé dans la méthode **pop()**

Exemple. suppression d'un élément du dictionnaire

```
1 stock={'Laptop': 15, 'Imprimante': 35, 'Tablette': 27, '
    Ipad':22}
2
3 # Suppression de l'élément "Imprimante": 35
4 stock.pop("Imprimante")
5 print(stock)
6 # affiche : {'Laptop': 15, 'Tablette': 27, 'Ipad':22}
```

Un dictionnaire est doté d'une autre méthode : **popitem()** qui permet de supprimer le dernier élément

Exemple. Suppression du dernier élément

```
1 stock={'Laptop': 15, 'Imprimante': 35, 'Tablette': 27, '
    Ipad':22}
2
3 # Suppression du dernier élément
4 stock.popitem()
5 print(stock)
6 # affiche : {'Laptop': 15, 'Imprimante': 35, 'Tablette':
    27}
```

1.10.3.4 Vider un dictionnaire

Un dictionnaire Python peut être vidé à l'aide de la méthode **clear()**

Exemple. vider un dictionnaire

```
1 stock={'Laptop': 15, 'Imprimante': 35, 'Tablette': 27, 'Ipad': 22}
2
3 # vider le dictionnaire
4 stock.clear()
5 print(stock)
6 # affiche un dictionnaire vide : {}
```

1.10.4 Récapitulatif des méthodes associées à un dictionnaire

Voici un récapitulatif des principales méthodes associées à un objet dictionnaire :

1. **clear()** : supprime tous les éléments du dictionnaire.
2. **copy()** : retourne une copie superficielle du dictionnaire.
3. **fromkeys(seq [, v])** : retourne un nouveau dictionnaire avec les clés de seq et une valeur égale à v (la valeur par défaut est None).
4. **get(key [, d])** : retourne la valeur de key. Si la clé ne existe pas, retourne d (la valeur par défaut est Aucune).
5. **items()** : retourne une nouvelle vue des éléments du dictionnaire (clé, valeur).
6. **keys()** : retourne une nouvelle vue des clés du dictionnaire.
7. **pop(key [, d])** : supprime l'élément avec key et renvoie sa valeur ou d si key n'est pas trouvé. Si d n'est pas fourni et que la clé est introuvable, soulève KeyError.
8. **popitem()** : supprimer et retourner un élément arbitraire (clé, valeur). Lève KeyError si le dictionnaire est vide.
9. **setdefault(key [, d])** : si key est dans le dictionnaire, retourne sa valeur. Sinon, insérez la clé avec la valeur d et renvoyez d (la valeur par défaut est Aucune).
10. **update([other])** : met à jour le dictionnaire avec les paires clé / valeur des autres clés existantes.
11. **values()** : retourne une nouvelle vue des valeurs du dictionnaire

1.11 Les ensembles Python (Python sets)

1.11.1 Définir un ensemble en Python

Un ensemble en Python (Python set) est une collection non ordonnée et non indexée. En Python, les ensembles sont écrits avec des accolades.

Exemple. Création d'un ensemble :

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 print(mySet)
```

Remarque 6. Les ensembles ne sont pas ordonnés, les éléments apparaitront donc dans un ordre aléatoire.

1.11.2 Accès aux éléments d'un ensemble Python

Vous ne pouvez pas **accéder** aux éléments d'un ensemble en faisant référence à un **index**, car les ensembles ne sont pas **ordonnés**, les éléments n'ont pas d'index. Mais vous pouvez **parcourir** les éléments de l'ensemble à l'aide d'une **boucle for** ou demander si une valeur spécifiée est présente dans un ensemble à l'aide du mot **clé in**.

Exemple. Affichage des éléments d'un ensemble

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 for x in mySet :
3     print(x)
```

Exemple. vérification d'appartenance d'un élément

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 print("Crayon" in mySet) # affiche : True
3 print("Cahier" in mySet) # affiche : False
```

1.11.3 Longueur ou cardinal d'un ensemble Python

Pour connaître la longueur (cardinal) d'un ensemble Python, on utilise la méthode **len()**

Exemple. longueur d'un ensemble python

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 cardinal = len(mySet)
3 print("card(mySet) = ", cardinal)
4 # affiche card(mySet) = 3
```

1.11.4 Opérations : ajouter, supprimer ou mettre à jour un ensemble Python

1.11.4.1 Ajouter un ou plusieurs éléments à un ensemble Python

- Pour **ajoutez un élément** à un ensemble Python, on utilise la méthode **add()** :

Exemple. Ajout d'un élément à l'ensemble

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.add("Cahier")
3 print(mySet)
```

- On peut aussi **ajouter plusieurs éléments** en même temps, mais cette fois ci avec la méthode **update()** :

Exemple. ajouter plusieurs éléments

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.update(["Cahier", "Cartable", "Trousse"])
3 print(mySet)
```

1.11.4.2 Supprimer un élément d'un ensemble Python

Pour supprimer un élément d'un ensemble Python, deux choix s'offrent à vous la méthode **remove()** ou la méthode **discard()**

Exemple. supprimer "Crayon" par la méthode **remove()**

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.remove("Crayon")
3 print(mySet) # affiche {'Gomme', 'Stylo'}
```

Remarque 7. Si l'élément à supprimer n'existe pas, **remove()** générera une erreur.

Exemple. supprimer "Crayon" par la méthode **discard()** :

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.discard("Crayon")
3 print(mySet) # affiche {'Gomme', 'Stylo'}
```

Remarque 8. Contrairement à la méthode **remove()**, la méthode **discard()** ne génère aucune erreur lorsque l'élément à supprimer n'existe pas ! L'instruction de suppression sera simplement ignorée !

Remarque 9. Vous pouvez également utiliser la méthode **pop()** pour supprimer un élément, mais cette méthode supprimera le dernier élément. Rappelez-vous que les ensembles ne sont pas ordonnés et vous ne saurez pas quel élément sera supprimé. La suppression est **totale**ment **aléatoire** !

1.11.4.3 Vider un ensemble Python

- Pour vider ensemble Python, on se sert de la méthode **clear()**

Exemple. vider un ensemble Python

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.clear()
3 print(mySet) # affiche set{} qui veut dire un ensemble vide
```

1.11.4.4 Supprimer un ensemble

Pour supprimer un ensemble Python, on utilise la commande **del**

Exemple. Supprimer un ensemble

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 del mySet
3 print(mySet)
4 # affiche le message d'erreur : builtins.NameError: name 'mySet' is not defined
```

1.11.5 Récapitulatif des méthodes associées à un ensemble Python

1. **add()** : ajoute un élément à l'ensemble
2. **clear()** : supprime tous les éléments de l'ensemble
3. **copy()** : retourne une copie de l'ensemble
4. **difference ()** : retourne un ensemble contenant la différence entre deux ensembles ou plus.
5. **difference_update()** : supprime les éléments de cet ensemble qui sont également inclus dans un autre ensemble spécifié
6. **discard()** : supprimer l'élément spécifié
7. **intersection()** : retourne un ensemble, qui est l'intersection de deux autres ensembles.

8. **intersection_update()** : supprime les éléments de cet ensemble qui ne sont pas présents dans d'autres ensembles spécifiés.
9. **isdisjoint()** : indique si deux ensembles ont une intersection ou non.
10. **issubset()** : indique si un autre jeu contient ce jeu ou non.
11. **issuperset()** : indique si cet ensemble contient un autre ensemble ou non.
12. **pop()** : supprime un élément de l'ensemble
13. **remove()** : supprime l'élément spécifié
14. **symmetric_difference()** : retourne un ensemble avec les différences symétriques de deux ensembles
15. **symmetric_difference_update()** : insère les différences symétriques de cet ensemble et d'un autre
16. **union()** : retourne un ensemble contenant l'union des ensembles
17. **update()** : met à jour l'ensemble avec l'union de cet ensemble et d'autres

1.12 Fonction Lambda En Python

1.12.1 A propos de la fonction lumbda

Dont le but de simplifier la syntaxe, Python nous donne la possibilité de ne pas déclarer une fonction de manière standard, c'est-à-dire en utilisant le **mot-clé def**. A cet effet ce type de fonctions est déclarée à l'aide du **mot-clé lambda**. Cependant, les **fonctions Lambda** peuvent accepter n'importe quel nombre d'arguments, mais elles ne peuvent renvoyer qu'une seule valeur sous la forme d'une expression.

1.12.2 Syntaxe de la fonction lumbda

La syntaxe d'une fonction **lumbda** est facile à comprendre à travers un exemple :

Exemple. Fonction lumbda : $x \mapsto x + 5$

```

1 # coding :-utf-8
2 # définir une fonction lumbda x —> x + 5
3 y = lambda x :x+5
4 # affichage du résultat
5 print (y)
6 print ("somme = " , y(7)) # affiche : somme = 12

```

1.12.3 Appliquer un filtre à une liste ou un tuple à l'aide d'une fonction lambda

La fonction **lambda** peut aussi être utilisée pour appliquer un filtre à une **liste**, un **tuple** ou un **dictionnaire**...

Exemple. Exemple filtrer les **éléments pairs** d'une liste de nombres :

```

1 # coding :-utf-8
2 # définir une liste
3 L = [13 , 16 , 22 , 31 , 17 , 46]
4
5 # filtrer les élément pair uniquement,t
6 list_even = list(filter(lambda x :(x%2 == 0) , L))
7
8 print(list_even) # affiche [16, 22, 46]
```

Exemple. Exemple filtrer les **personnes majeurs** dans un dictionnaire :

```

1 # coding :-utf-8
2 # définir un dictionnaire des personnes avec leurs âges
3 d = {'Ahmed' : 15 , 'Robert' : 18 , 'Nathalie' : 19 , '
    Cecilia' : 13 , 'David' : 14 }
4
5 # filtrer les personnes majeurs
6 person_major = list(filter(lambda x :(d[x] >= 18) , d))
7
8 print(person_major) # affiche ['Robert', 'Nathalie']
```

1.12.4 Transformer une liste ou un tuple avec la fonction lambda et la fonction map()

La fonction **map()** en Python accepte une fonction et une liste. Il donne une nouvelle liste qui contient tous les éléments modifiés renvoyés par la fonction pour chaque élément.

Exemple. liste des carrées d'une liste de nombre avec les fonctions **lambda** et **map()** :

```

1 #programme qui renvoie la liste des carrées des éléments d
  'une liste
2 L = [11 , 3 , 7 , 2 , 5 , 9]
3 liste_carrees = list(map(lambda x :x**2 , L))
4
5 # afficher la liste des carrées
6 print(liste_carrees) # affiche [121, 9, 49, 4, 25, 81]
```

1.13 Compréhension des listes en Python

1.13.1 Qu'est ce que la compréhension des listes ?

La compréhension des listes est considéré comme un moyen élégant de définir, de créer une liste en Python et consiste à placer entre des crochets, une expression suivie d'une ou plusieurs structures **for**, **if**, **while**... Il s'agit d'une méthode efficace à la fois en termes de calcul, en termes de codage, d'espace et de temps.

1.13.2 Syntaxe

[expression(**item**) condition(**item**)]

Exemple. .

```
1 # -*- coding: utf-8 -*-
2 s = "Python" # Création d'une liste formée des caractères
   de la chaîne s
3 L = [item for item in s]
4 print(L) # affiche ['P', 'y', 't', 'h', 'o', 'n']
```

Chapitre 2

Programmation orientée objet POO en Python

2.1 Le concept de POO en Python

La **programmation orientée objet**, ou **POO**, est un **paradigme de programmation** qui permet de structurer les programmes de manière à ce que les propriétés et les comportements soient regroupés dans des objets à part.

Par exemple, un objet peut représenter une personne avec un nom, un âge, une adresse, etc., avec des comportements tels que marcher, parler, respirer et courir.

En d'autres termes, la programmation orientée objet est une approche permettant de modéliser des éléments concrets du monde réel tels que les voitures, ainsi que des relations entre des entités telles que les entreprises et les employés, les étudiants et les enseignants, etc. La modélisation POO modélise des entités réelles sous la forme d'objets logiciels certaines données qui leur sont associées et peuvent remplir certaines fonctions.

La **programmation orienté objet** est un type de programmation basée sur la création des **classes** et des **objets** via une méthode appelée **instanciation**. Une classe est un prototype (modèle) codé en un langage de programmation dont le but de créer des objets dotés d'un ensemble de méthodes et attributs qui caractérisent n'importe quel objet de la classe. Les attributs sont des types de données (variables de classe et variables d'instance) et des méthodes, accessibles via la concaténation par points. En programmation orientée objet, la déclaration d'une classe regroupe des méthodes et propriétés (attributs) communs à un ensemble d'objets. Ainsi on

pourrait dire qu'une classe représente une catégorie d'objets. Elle apparaît aussi comme une usine permettant de créer des objets ayant un ensemble d'attributs et méthodes communes.

Depuis sa création, **Python** est un langage de **programmation orienté objet**. Pour cette raison, la création et l'utilisation de classes et d'objets en Python est une opération assez simple. Ce cours vous aidera à apprendre étape par étape l'usage de la programmation orienté objet en Python.

2.2 Terminologie de la POO

1. **Classe** - modèle de code abstrait qui représente et caractérise une catégorie d'objets ayant des propriétés communes (**attributs et méthodes**). La création d'un objet à partir d'une classe est appelé **instanciation**. L'objet crée à partir d'une classe est appelé **instance** ou **objet d'instance** (**instance object** en anglais)
2. **Variable de classe** - Variable partagée par toutes les instances d'une classe. Les variables de classe sont définies dans une classe mais en dehors de toute méthode de la classe. Les variables de classe ne sont pas utilisées aussi souvent que les variables d'instance.
3. **Data member** - Une variable de classe ou une variable d'instance qui contient les données associées à une classe et à ses objets.
4. **Surcharge de fonction (Function overloading)** - Affectation de plusieurs comportements à une fonction particulière. L'opération effectuée varie en fonction des types d'objets ou d'arguments impliqués.
5. **Variable d'instance** - Variable définie dans une méthode et n'appartenant qu'à l'instance actuelle d'une classe.
6. **Héritage** - Transfert des caractéristiques d'une classe à d'autres classes qui en sont dérivées.
7. **Instance** - Un objet crée à partir d'une certaine classe. Un objet obj qui appartient à une classe Circle, par exemple, est une instance de la classe Circle.
8. **Instanciation** - Création d'une instance d'une classe.
9. **Méthode** - Type particulier de fonction défini dans une définition de classe.
10. **Object** - Une instance unique d'une structure de données définie par sa classe. Un objet comprend à la fois des membres de données (variables de classe et variables d'instance) et des méthodes.

11. **Surcharge d'opérateur** - Affectation de plusieurs fonctions à un opérateur particulier.

2.3 Les classes en Python

Pour créer une classe en Python, on utilise l'instruction :

```
1 class nom_de_la_classe
```

On crée ensuite une méthode qui permet de construire les objets, appelé constructeur via l'instruction :

```
1 def __init__( self ) :
```

Exemple. classe **Personne**

```
1 class Personne :
2     def __init__( self ,nom,age ) :
3         self.nom = nom
4         self.age = age
5 P = Personne("Albert",27)
6 print("Le nom de la prsonne est : " , P.nom)
7 print("L'age de la personne est : " , P.age, " ans")
8 # affiche : Le nom de la prsonne est : Albert
9 #          L'age de la personne est : 27 ans
```

Exemple. classe **Rectangle**

```
1 class Rectangle :
2     def __init__( self ,L,l ) :
3         self.Longueur=L
4         self.Largeur=l
5 monRectangle=Rectangle(7,5)
6 print("La longueur de mon rectangle est : " ,monRectangle.
    Longueur)
7 print("La largeur de mon rectangle est : " ,monRectangle.
    Largeur)
```

Ce qui affiche à l'exécution :

La longueur de mon rectangle est : 7

La largeur de mon rectangle est : 5

On peut aussi améliorer la classe en ajoutant des méthodes permettant d'effectuer différentes tâches.

2.4 Les méthodes d'instances en Python

Définition 1. Une méthode d'instance est une méthode ou procédure nommée au sein de la classe, permettant de définir des propriétés ou comportements des objets d'instances. Une méthode d'instance possède un ou plusieurs paramètre, le premier nommé **self** est obligatoire !

Exemple. ajout de méthode d'instance qui calcule la **surface** du **rectangle**

```

1  #coding : utf-8
2  class Rectangle :
3      def __init__(self ,L,l) :
4          self.Longueur=L
5          self.Largeur=l
6
7      # méthode qui calcule la surface
8      def surface(self) :
9          return self.Longueur*self.Largeur
10
11 # création d'un rectangle de longueur 7 et de largeur 5
12 monRectangle = Rectangle(7,5)
13 print("La surface de mon rectangle est : ", monRectangle.
    surface())

```

Ce qui affiche après exécution : *La surface de mon rectangle est : 35*

2.5 Les méthodes de classes en Python

Une méthode de classe en Python est une méthode qui fonctionne au sein de la classe où elle a été créée et elle est accessible par un objet d'instance ou directement en utilisant le nom de la classe sans faire aucune instantiation ! Une méthode de classe est caractérisée par :

1. Elle fonctionne à l'intérieur de la classe où elle a été et est accessible via une instance objet ou en utilisant le nom de la classe !
2. Une méthode de classe est décorée par **@classmethod**
3. Une méthode de classe possède un premier paramètre obligatoire nommé **cls**

2.6 Attributs d'instances et attributs de classes

Un attribut d'instance est un attribut qui fonctionne avec un objet d'instance tandis qu'un attribut de classe est un attribut qui fonctionne avec la classe

Exemple 1. attributs d'instances & attributs de classes

```

1 class Student :
2     name = 'Aladin' # attribut de classe
3     def __init__(self) :
4         self.age = 30 # attribut d'instance
5
6     @classmethod
7     def displayName(cls) :
8         print("Le nom de l'étudiant est : " , cls.name)
9
10    print(Student.displayName())# affiche : Le nom de l'
        etudiant est : Aladin
11    Stud = Student()
12    print(Stud.displayName())# affiche : Le nom de l'étudiant
        est : Aladin

```

2.7 Les méthodes statiques

Une méthode statique est une méthode de classe ayant la propriété d'être exécutée sans passer par l'instanciation

Exemple. méthode statique

```

1 class myClass :
2     def __init__(self) :
3         pass
4     # création d'une méthode statique
5     @staticmethod
6     def myStaticMethod() :
7         print("Voici un exemple de méthode statique en
            Python")
8
9    myClass.myStaticMethod()

```

2.8 Héritage en Python

Pour éviter de recopier le code d'une classe, on utilise la méthode d'héritage. La méthode d'héritage consiste à créer à partir d'une classe mère une autre classe appelé classe fille qui hérite toutes les méthodes et propriétés de la classe mère. Pour simplifier l'acquisition pour les apprenants débutant, nous allons traiter ce concept sur un exemple simple :

Classe mère :

```

1 #-*- coding : utf-8 -*-
2 class Personne :

```



```

3     def __init__( self ,nom, age ) :
4         self.nom = nom
5         self.age=age

```

Nous venons de définir une **classe** **Personne** dont les attributs sont **nom** et **age**. Nous allons maintenant créer une **classe fille** nommée **Student** qui **hérite** les mêmes **méthodes** et **propriétés** de la classes mère **Personne**. La syntaxe générale de l'**héritage** se fait grâce à la **commande** :

```

1 class classe_fille (classe_mère)

```

Qui veut dire que la classe **classe_fille** hérite de la calsse **classe_mère**.

Exemple pour notre cas de la **classe fille Student** qui hérite de la **classe mère Personne** :

```

1 class Student(Personne) :

```

L'héritage des attributs **nom** et **age** se fait via la commande :

```

1 Personne.__init__( self ,nom, age)

```

Code de la **classe fille Student** :

```

1  -*- coding : utf-8 -*-
2  class Student(Personne) :
3      # définition des attributs des attributs
4      def __init__( self ,nom,age , filiere ) :
5          # héritage des attributs depuis la classe mère
          Personne
6          Personne.__init__( self ,nom,age)
7          # ajout d'un nouvel attribut filiere à la classe
          fille
8          self.filiere = filiere

```

Exemple. (complet)

```

1  -*- coding : utf-8 -*-
2  class Personne :
3      def __init__( self ,nom, age ) :
4          self.nom = nom
5          self.age=age
6  # La classe fille Student hérite de la classe mère
          Personne
7  class Student(Personne) :
8      # définition des attributs des attributs
9      def __init__( self ,nom,age , filiere ) :
10         # héritage des attributs depuis la classe mère
            Personne
11         Personne.__init__( self ,nom, age)

```

```

12         # ajout d'un nouvel attribut filiere à la classe
        fille
13         self.filiere = filiere
14 Stud = Student("Albert",27,"math")
15 print("Le nom de l'étudiant est : " ,Stud.nom)
16 print("L'age de l'étudiant est : " ,Stud.age)
17 print("La filière de l'étudiant est : " ,Stud.filiere)

```

Ce qui affiche après exécution :

Le nom de l'étudiant est : Albert

L'age de l'étudiant est : 27

La filière de l'étudiant est : math

2.9 Héritage multiple

Exemple. héritage multiple

```

1 class Calcul1 :
2     def Somme(self ,a,b) :
3         return a + b
4
5 class Calcul2 :
6     def Multiplication(self ,a,b) :
7         return a*b
8
9 # heritage multiple
10 class Calcul3(Calcul1 , Calcul2) :
11     def division(self ,a,b) :
12         return a/b
13
14 Calcul = Calcul3()
15 print(Calcul.Somme(7 , 5)) # affiche 12
16 print(Calcul.Multiplication(10 , 5)) # affiche 50
17 print(Calcul.division(8 , 3)) # affiche 2.666666

```

2.10 Surcharge de méthodes (overloading)

La surcharge en programmation orientée objet est méthode permettant de définir plusieurs fois une même méthode avec des arguments différents. Le compilateur choisit la méthode qui doit être appelée en fonction du nombre et du type des arguments .

Contrairement aux autres langages objet Java, C++, python ne prend pas en charge la surcharge de méthode par défaut. Mais il existe différentes façons d'obtenir une surcharge de méthode en Python.

Le problème avec la surcharge de méthodes en Python est que nous pouvons surcharger les méthodes mais ne pouvons utiliser que la dernière méthode définie.

Exemple. Surcharge de méthodes

```

1 class Calcul :
2
3     def __init__( self ) :
4         pass
5
6     def somme( self , x , y ) :
7         return x + y
8
9     def somme( self , a , b , c ) :
10        return a + b + c
11
12 C = Calcul()
13 print( "La somme est 2 + 3 + 4 = " , C.somme(2 , 3 , 4))
14 # affiche : La somme est 2 + 3 + 4 = 9

```

Remarque 10. L'opération :

```

1 C.somme(2 , 3)

```

ne peut être exécuter ! Néanmoins il y a une possibilité de faire fonctionner correctement la surcharge de méthodes en Python via le module **multipledispatch** qu'on peut installer facilement via l'utilitaire **pip** :

```

1 pip install multipledispatch

```

Exemple. Surcharge de méthodes via **multipledispatch**

```

1 from multipledispatch import dispatch
2 class Calcul :
3
4     def __init__( self ) :
5         pass
6
7     @dispatch(int , int)
8     def somme( self , x , y ) :
9         return x + y
10
11    @dispatch(int , int , int)
12    def somme( self , a , b , c ) :
13        return a + b + c
14
15    @dispatch(float , float)
16    def somme( self , a , b ) :
17        return a + b

```

```

18
19 C = Calcul()
20 print("La somme de 2 et 3 est : " , C.somme(2 , 3))
21 print("La somme de 2.5 et 3.2 est : " , C.somme(2.5 , 3.2)
    )
22 print("La somme de 2 , 3 et 4 est : " , C.somme(2 , 3 ,
    4))

```

Affiche à la sortie :

La somme de 2 et 3 est : 5

La somme de 2.5 et 3.2 est : 5.7

La somme de 2 , 3 et 4 est : 9

2.11 Polymorphisme et redéfinition de méthodes (overriding methods)

Linguistiquement, le mot **polymorphisme** signifie plusieurs formes. En Python, le **polymorphisme** sert à définir une méthodes dans la **classe enfant** avec le **même nom** qu'une méthodes de la **classe mère**. Le principe de l'**héritage polymorphe** est d'offrir la possibilité à une classe enfant d'hériter une méthode de la **classe mère avec modification** selon le besoin de la classe enfant. Exemple pour une **classe Personne** on peut définir une **méthode afficher()** qui affiche les **attributs** d'un objet : **nom, prénom, age...** et dans une autre **classe Etudiant** qui **hérite** de la **classe Personne** on aura besoin de la même méthode **afficher()** mais cette fois elle affiche des informations supplémentaires spécifique à la classe enfant comme **section, niveau d'études...**

Exemple. (redéfinition de méthodes)

```

1 class Personne :
2     def __init__(self , name , age) :
3         self.name = name
4         self.age = age
5     def afficher(self) :
6         print("le nom est : " , self.name ,
7             " l'age est : " , self.age)
8
9 # Heritage de la classe parent
10 class Student(Personne) :
11     def __init__(self , name , age , section) :
12         Personne.__init__(self , name , age)
13         self.section = section
14
15 # modification de la methode afficher

```

```

16     def afficher(self):
17         print("le nom est : " , self.name ,
18             " l'age est : " , self.age,
19             "la section de l'étudiant est : " , self.
20             section)
21
22 Stud = Student("Majid" , 23 , "informatique")
23 print(Stud.afficher())
24 # la sortie est : e nom est : Majid l'age est : 23 la
    section de l'étudiant est : informatique

```

2.12 Les classes abstraites en Python

Les **classes abstraites** sont des classes qui contiennent une ou plusieurs **méthodes abstraites**. Une **méthode abstraite** est une méthode déclarée, mais qui **ne contient aucune implémentation**. Les **classes abstraites ne peuvent pas être instanciées** et nécessitent des sous-classes pour fournir des implémentations pour les méthodes abstraites. afin de pouvoir utiliser les classes abstraites, il faut au préalable importer la classe **ABC** depuis le module **abc** :

```

1 from abc import ABC , abstractmethod

```

Exemple. (classe abstraite)

```

1 from abc import ABC , abstractmethod
2
3 # define abstract class
4 class Polygon(ABC):
5
6     # define abstract method
7     @abstractmethod
8     def sidesNumber(self):
9         pass
10
11 class Triangle(Polygon):
12     def sidesNumber(self):
13         print("Un triangle admet 3 cotes")
14
15 class Rectangle(Polygon):
16     def sidesNumber(self):
17         print("Un rectangle admet 4 cotes")
18
19 # Instanciation
20 T = Triangle()
21 T.sidesNumber()# affiche : Un triangle admet 3 cotes

```

```

22
23 R = Rectangle()
24 R.sidesNumber() # affiche : Un rectangle admet 4 cotes

```

2.13 Les interfaces en Python

Une interface est un type de référence regroupant une collection de méthodes abstraites tout en exigeant un certain nombre de règles :

1. **Une interface** établit un **contrat** entre une classe et l'interface qu'elle implémente.
2. **Une interface** ne contient que des **prototypes** de **méthodes non-implémentées**.
3. La classe qui implémente l'interface doit fournir une implémentation pour chacune des méthodes de l'interface.
4. En générale on parle d'implémenter une interface (et non hériter d'une interface), mais en Python il s'agit d'un type d'héritage.

Python est doté d'une bibliothèque nommée **interface** permettant de déclarer des interfaces et pour affirmer statiquement que les classes implémentent ces interfaces. Il fournit une sémantique plus stricte que le module `abc` intégré de Python et vise à produire des messages d'erreur d'une utilité exceptionnelle lorsque les interfaces ne sont pas satisfaites.

Afin de pouvoir utiliser les interfaces en Python, il faut installer le module `interface` via l'utilitaire `pip` :

```

1 pip install interface

```

```

1 from interface import implements, Interface
2
3 class MyInterface(Interface):
4
5     def somme(self, x, y):
6         pass
7
8 class MyClass(implements(MyInterface)):
9
10    def somme(self, x, y):
11        return x + y
12
13 My = MyClass()
14 print(My.somme(3, 5)) # affiche 8

```

Pour plus de détails sur les interfaces Python veuillez consulter la documentation officielle : <https://interface.readthedocs.io/en/latest/>

Chapitre 3

Les modules en Python

3.1 Introduction

Un module en Python est simplement un fichier constitué de code Python qu'on peut appeler et utiliser son code sans avoir besoin de le recopier. Un module peut contenir des fonctions, des classes, des variables... Un module vous permet d'organiser logiquement votre code Python. Le regroupement de code associé dans un module rend le code plus facile à comprendre et à utiliser.

Le langage Python trois sortes de modules :

1. Les modules standard intégrés automatiquement par Python, fournissant un accès à des opérations qui ne font pas partie du noyau du langage mais qui sont néanmoins intégrées.
2. Les modules externes développés par des développeurs bénévoles.
3. Les modules qu'on peut développer soi mêmes.

3.2 Créer votre propre module

Nous allons essayer de créer notre propre module Python nommée **myModule** :

1. On crée un fichier nommé **myModule.py**
2. On introduit un code de quelques fonctions simples sur 1 fichier **myModule.py** par exemple :

```
1 def somme(x, y) :  
2     return x + y
```

```

3
4 def division(x,y) :
5     return x/y

```

- On crée ensuite un fichier python pour tester le module par exemple **testModule.py** dans le même répertoire que le fichier **myModule.py** (les deux fichiers **myModule.py** et **testModule.py** peuvent être placés sur des **répertoires différents** à condition de préciser le **chemin du fichiers myModule.py** lors de son **importation**)
- Sur le fichier **testModule.py** tapons le code :

```

1 # On importe la totalité du module
2 from myModule import *
3
4 # On peut maintenant utiliser les fonction du module
5 print("la somme de 7 et 8 est :",somme(7,8))
6 print("la division de 12 par 3 est :", division
    (12,3))

```

Remarque 11. Pour utiliser les fonctions d’un module, il n’est pas nécessaire d’importer la totalité du module, mais il suffit d’importer juste les fonctions dont on a besoin. Par exemple si on a besoin d’utiliser uniquement la fonction `somme()`, on import juste cette dernière.

Exemple. importation partielle du module :

```

1 # On importe la fonction somme() du module
2 from myModule import somme
3
4 # On peut maintenant utiliser les fonction du module :
5 print("la somme de 7 et de 8 est :",somme(7,8))

```

3.3 les modules standards en Python

Les modules standards en Python sont très nombreux dont on peu citer les plus utilisés :

- Le module cgi** (Common Gateway Interface en anglais) qui veut dire “Interface de Passerelle Commune permet l’exécution des programmes Python sur des serveurs HTTP
- Le module calendar** : contient les fonctions de calendrier
- Le module datetime** : fournit les classes principales pour manipuler des dates et des heures

4. **Le module json** : permet la gestion des données au format JSON
5. **Le module math** : contient un ensemble de fonctions permettant de réaliser des calculs mathématiques
6. **Le module os** : fournit une manière portable d'utiliser les fonctionnalités dépendantes du système d'exploitation
7. **Le module pickle** : permet de sérialiser des objets Python
8. **Le module plateforme** : permet de fournir diverses informations système.
9. **Le module profile** : contient les programmes qui permettent l'analyse et l'exécution des fonctions.
10. Le module `pyinstaller` : permet de créer un exécutable pour un script python.
11. **Le module random** : implémente des générateurs de nombres pseudo-aléatoires pour différentes distributions
12. **Le module re** : fournit des opérations sur les expressions rationnelles similaires à celles que l'on trouve dans Perl
13. **Le module statistics** : fournit des outils permettant d'effectuer des analyses statistiques
14. **Le module socket** : fournit un accès à l'interface sockets qui correspond à un ensemble normalisé de fonctions de communication
15. **Le module sys** : permet l'accès à certaines fonctions et variables système
16. **Le module time** : utilisé pour interagir avec le temps système
17. **Les modules urllib.request et urllib.parse** permettent d'ouvrir, de lire et d'analyser des URLs.
18. **Le module virtualenv** : permet de créer un environnement virtuel.

Chapitre 4

Exemple d'usage de quelques modules standards

4.1 Le module os

Le module `os` en python est utilisé pour gérer tout ce qui a un rapport avec le système d'exploitation, et vu qu'il soit aussi étroitement lié aux fichiers, nous allons le traiter en détail dans le chapitre suivant (*les fichiers en Python*)

4.2 Le module statistics

Ce module fournit des fonctions permettant de faire des calculs statistiques mathématiques à partir des données numériques (valeurs réelles).

Le module n'est pas destiné à concurrencer des bibliothèques tierces telles que **NumPy**, **SciPy** ou des ensembles de statistiques propriétaires complets destinés aux statisticiens professionnels.

Principales fonctions du module statistics :

1. **mean ()** : moyenne arithmétique (moyenne) des données.
2. **fmean ()** : moyenne arithmétique rapide en virgule flottante.
3. **geometric_mean ()** : moyenne géométrique des données.
4. **harmonic_mean ()** : moyenne harmonique des données.

5. **médiane ()** : médiane (valeur moyenne) des données.
6. **median_low ()** : faible médiane des données.
7. **median_high ()** : médiane élevée des données.
8. **median_grouped ()** : médiane, ou 50e centile, des données groupées.
9. **mode ()** : mode unique (valeur la plus courante) de données discrètes ou nominales.
10. **multimode ()** : liste des modes (valeurs les plus courantes) de données discrètes ou nominales.
11. **quantiles ()** : divisez les données en intervalles avec une probabilité égale.
12. **pstdev ()** : écart type des données de population.
13. **pvariance ()** : Variance des données de la population.
14. **stdev ()** : exemple d'écart-type de données.
15. **variance ()** : Variance de l'échantillon de données.

Exemple. Calcul de la moyenne

```
1 import statistics as st
2 valeurs =[17,21,22,18,19,17,21]
3 print("La moyenne des valeurs est : " , st.mean(valeurs))
```

Exemple. Calcul de la médiane

```
1 import statistics as st
2 valeurs =[17,21,22,18,19,17,21]
3 print("La mediane des valeurs est : " , st.median(valeurs))
```

Exemple. Calcul de la variance

```
1 import statistics as st
2 valeurs =[17,21,22,18,19,17,21]
3 print("La variance est : " , st.variance(valeurs))
```

Exemple. Calcul du mode d'une série statistique

```
1 from statistics import mode
2 valeurs =[21 , 7 , 2 , 7 , 3 , 7 , 5]
3 print("Le mode des valeurs est : " , mode(valeurs))
```

4.3 le module **virtualenv**

4.3.1 A propos du module **virtualenv**

Python est doté d'un module nommée **virtualenv** permettant de créer un **environnement virtuel python**, c'est-à-dire une copie de travail isolée de Python qui vous permet de travailler sur un projet spécifique sans affecter d'autres projets. Donc, fondamentalement, c'est un outil qui permet plusieurs installations côte à côte de Python, c.a.d une installation propre pour chaque projet.

4.3.2 Création d'un environnement **virtualenv** sous Windows

Pour créer un **environnement virtuel** sous Windows, il suffit de suivre les étapes suivantes :

1. Lancez la commande **cmd**
2. Naviguer jusqu'au répertoire de votre projet via la commande **'cd'**
3. Installez **virtualenv** en tapant la commande : **'pip install virtualenv'**
4. Créer un **virtualenv** nommez le à titre d'exemple **my_venv** via la commande : **'virtualenv my_venv'**
5. Maintenant, si vous êtes dans le même répertoire, activer votre environnement virtuel en tapant : **'my_venv\Scripts\activate.bat'**
6. Vous pouvez **désactiver** votre **virtualenv** en tapant la commande **cmd** : **'deactivate'**

4.4 Le module **PyInstaller** : Transformer un script Python en un exécutable Windows

4.4.1 Les outils de création d'un exécutable Windows

Pour convertir un **script Python** en un **exécutable Windows**, rien de plus simple ! Il existe de nombreux outils disponibles pour réaliser de telle tâche ! Nous pouvons citer à titre d'exemple :

1. **PyInstaller**
2. **cx_Freeze**
3. **py2exe**

L'outil que nous allons adopter dans cet article est **PyInstaller** ! C'est un outil pratique spécialement conçu pour créer un fichier **.exe** pour Windows avec une seule commande !

4.4.2 Installation

Installation est pratiquement très simple, tout ce que vous avez à faire est d'exécuter la commande suivante (si vous avez déjà installé python avec l'**utilitaire pip**) :

```
1 pip install pyinstaller
```

Si vous êtes sous Windows, vous devez aussi installer l'outil **pywin32** :

```
1 pip install pywin32
```

4.4.3 Création de l'exécutable

Maintenant que vous avez installé **PyInstaller**, il vous suffit de répertorier le script python que vous souhaitez convertir en exécutable, d'ouvrir ensuite la **commande cmd**, de naviguer jusqu'au répertoire de votre script à l'aide de la commande **cd** (change directory) et de lancer la commande :

```
1 pyinstaller --onefile <votre_script>.py
```

4.5 Le module math

4.5.1 A propos du module math Python

Le module mathématique Python fournit les fonctions mathématiques les plus populaires, qui incluent les fonctions trigonométriques, les fonctions de représentation, les fonctions logarithmiques, etc. En outre, il définit également les constantes mathématiques comme le **nombre pi** et le **nombre d'Euler e**, etc.

- **Le nombre pi** : est une constante mathématique bien connue et définie comme le rapport du périmètre par rapport au diamètre d'un cercle. Sa valeur approximative est **$\pi = 3,141592653589793$** .
- **Le nombre d'Euler e** : est défini comme la base du logarithmique népérien, et sa valeur est **$e = 2,718281828459045$** .

4.5.2 Les fonctions représentant des nombres en Python

Python fournit différentes fonctions qui sont utilisées pour représenter des nombres sous différentes formes, par exemple :

1. **ceil(x)** : renvoie la valeur plafond qui est la plus petite valeur entière, supérieure ou égale au nombre x.
2. **copysign(x, y)** : renvoie le nombre de x et copie le signe de y dans x.
3. **fabs(x)** : renvoie la valeur absolue de x.
4. **factorial(x)** : renvoie la factorielle de x où $x \geq 0$
5. **floor(x)** : renvoie la partie entière d'un nombre x.
6. **fsum(itérable)** : renvoie la somme des éléments d'un objet itérable
7. **gcd(x, y)** : renvoie le plus grand diviseur commun de x et y.
8. **isfinite(x)** : vérifie si x n'est ni un infini ni un NAN.
9. **isinf(x)** : vérifie si x est infini
10. **isnan(s)** : vérifie si s n'est pas un nombre
11. **remainder(x,y)** : donne le reste après avoir divisé x par y.

Exemple.

```
1 import math
2 print(math.fabs(-5)) # affiche 5.0
3 print(math.floor(3.7)) # affiche 3
4 print(math.gcd(8,12)) # affiche 4
5 print(math.remainder(17,6)) # affiche -1.0
6 print(math.remainder(17,5)) # affiche 2.0
7 print(math.factorial(4)) # affiche 24
```

4.5.3 Fonctions de conversion trigonométrique et angulaire

Ces fonctions sont utilisées pour calculer différentes opérations trigonométriques :

1. **sin(x)** : renvoie le sinus de x en radians
2. **cos(x)** : renvoie le cosinus de x en radians
3. **tan(x)** : renvoie la tangente de x en radians
4. **asin (x)** : retourne l'inverse du sinus, de même nous avons **acos**, **atan** aussi

5. **degrees(x)** : convertit l'angle x du radian en degrés
6. **radians(x)** : convertit l'angle x des degrés en radian

Exemple.

```
1 import math
2 print(math.sin(math.pi/3)) # affiche 0.8660254037844386
3 print(math.tan(math.pi/4)) # affiche 0.9999999999999999
4 print(math.degrees(math.pi/2)) # affiche 90.0
```

4.5.4 Fonctions exponentielle & logarithmiques

1. **pow(x, y)** : renvoie x à la puissance y ie : x^y
2. **sqrt(x)** : renvoie la racine carrée de x
3. **exp(x)** : renvoie l'exponentielle de x .
4. **log(x [, base])** : renvoie le logarithme de x où la base est donnée en argument. La base par défaut est e
5. **log2(x)** : renvoie le logarithme de x , où la base est 2.
6. **log10(x)** : renvoie le logarithme de x , où la base est 10.

Exemple. .

```
1 import math
2 print(math.pow(2,3)) # affiche 8.0
3 print(math.exp(1)) # affiche 2.718281828459045
4 print(math.log(2)) # affiche 0.6931471805599453
```

4.6 Le module random

4.6.1 A propos du module random

Le module **random** en Python est un module intégré de Python permettant d'effectuer des actions **aléatoires** telles que la génération de nombres aléatoires, l'impression aléatoire d'une valeur pour une liste ou une chaîne, etc.

4.6.2 Les méthodes associées au module random

1. **seed()** : Initialise le générateur de nombres aléatoires
2. **getstate()** : renvoie l'état interne actuel du générateur de nombres aléatoires

3. **setstate()** : restaure l'état interne du générateur de nombres aléatoires
4. **getrandbits()** : renvoie un nombre représentant les bits aléatoires
5. **randrange()** : renvoie un nombre aléatoire entre la plage donnée
6. **randint()** : renvoie un nombre aléatoire entre la plage donnée
7. **choice()** : renvoie un élément aléatoire de la séquence donnée
8. **choices()** : renvoie une liste avec une sélection aléatoire dans la séquence donnée
9. **shuffle()** : Prend une séquence et renvoie la séquence dans un ordre aléatoire
10. **sample()** : renvoie un échantillon donné d'une séquence
11. **random()** : renvoie un nombre flottant aléatoire entre 0 et 1
12. **uniform()** : renvoie un nombre flottant aléatoire entre deux paramètres donnés
13. **triangular()** : renvoie un nombre flottant aléatoire entre deux paramètres donnés, vous pouvez également définir un paramètre de mode pour spécifier le point médian entre les deux autres paramètres
14. **betavariate()** : renvoie un nombre flottant aléatoire entre 0 et 1 basé sur la distribution Beta (utilisée dans les statistiques)
15. **expovariate()** : renvoie un nombre flottant aléatoire basé sur la distribution exponentielle (utilisée dans les statistiques)
16. **gammavariate()** : renvoie un nombre flottant aléatoire basé sur la distribution Gamma (utilisée dans les statistiques)
17. **gauss()** : renvoie un nombre flottant aléatoire basé sur la distribution gaussienne (utilisée dans les théories des probabilités)
18. **lognormvariate()** : renvoie un nombre flottant aléatoire basé sur une distribution log-normale (utilisée dans les théories des probabilités)
19. **normalvariate()** : renvoie un nombre flottant aléatoire basé sur la distribution normale (utilisé dans les théories des probabilités)
20. **vonmisesvariate()** : renvoie un nombre flottant aléatoire basé sur la distribution de von Mises (utilisée dans les statistiques directionnelles)
21. **paretovariate()** : renvoie un nombre flottant aléatoire basé sur la distribution de Pareto (utilisée en théorie des probabilités)

22. **weibullvariate()** : renvoie un nombre flottant aléatoire basé sur la distribution de Weibull (utilisée dans les statistiques)

Exemple. (afficher un nombre aléatoire sur plage donnée)

```
1 import random
2 # afficher un nombre aléatoire entre 5 et 13
3 print(random.randrange(5, 13))
```

Exemple. (extraire une sous liste aléatoire)

```
1 import random
2 # extraire aléatoirement une sous liste de trois éléments
3 data = ["Java", "Python", "PHP", "Django"]
4 print(random.sample(data, 3))
```

Exemple. (changer aléatoirement l'ordre des éléments d'une liste)

```
1 import random
2 # changer aléatoirement l'ordre
3 data = ["Java", "Python", "PHP", "Django"]
4 random.shuffle(data)
5 print(data)
```

4.7 Le module collection

doc officielle : <https://docs.python.org/fr/3/library/queue.html>

Chapitre 5

Les fichiers en Python

5.1 Le module os

Le module `os` est fourni par Python dont le but d'interagir avec le système d'exploitation, il permet ainsi de gérer l'arborescence des fichiers, de fournir des informations sur le système d'exploitation processus, variables systèmes...

Le module `os` peut être chargé simplement avec la commande : **`import os`**

5.1.1 La méthode `os.getlogin()`

`os.getlogin()` : renvoie le nom d'utilisateur courant.

Exemple. programme Python qui renvoie le nom d'utilisateur :

```
1 import os
2 user = os.getlogin()
3 print(user) # imprime le nom d'utilisateur
```

5.1.2 La méthode `os.mkdir()`

`os.mkdir(chemin)` : crée un répertoire correspondant au chemin spécifié.

Exemple. création d'un dossier à la racine du disque C :

```
1 import os
2 os.mkdir("c:/myFolder") # crée un dossier nommé myFolder
   sur le disque C:\
```

5.1.3 La méthode `os.getcwd()`

`os.getcwd()` : renvoie le répertoire actuel sous forme de chaîne de caractères.

```
1 import os
2 rep_actuel = os.getcwd()
3 print(rep_actuel) # renvoie le répertoire actuel
```

5.1.4 La méthode `os.path`

Afin de pouvoir utiliser la méthode `os.path`, il faut préalablement importer le module `pathlib`. Le module `pathlib` est module doté d'une interface orientée objet inclus dans python depuis la version 3.4 doté de méthodes très intuitives permettant d'interagir avec le système de fichiers d'une façon simple et conviviale.

5.1.4.1 Tester si un répertoire existe avec la méthode `os.path.exists()`

La méthode `os.path.exists()` permet de tester si un répertoire existe ou non

Exemple. tester l'existence d'un répertoire

```
1 import os
2 from pathlib import Path
3 print(os.path.exists("c:/users/"))
4 #affiche True
5 #-----
6 # On peut aussi utiliser
7 print(not os.path.exists("c:/users/"))
8 #affiche False
```

5.1.4.2 Tester si un chemin est un répertoire ou un fichier avec les méthodes `is_dir()` et `is_file()`

Pour tester la nature d'un chemin s'il s'agit d'un répertoire ou un fichier on utilise les méthodes `is_dir()` et `is_file()`

Exemple. ...

```
1 import os
2 from pathlib import Path
3 myDirectory = "C:/Users"
4 p = Path(myDirectory)
5 print(p.is_dir()) # affiche True
6 print(p.is_file()) # affiche False
```

Exemple. ...

```
1 import os
2 from pathlib import Path
3 myDirectory = "C:/Windows/system.ini"
4 p = Path(myDirectory)
5 print(p.is_dir()) # affiche False
6 print(p.is_file()) # affiche True
```

5.1.4.3 Détermination du chemin absolu à l'aide de la méthode `absolute()`

La méthode `absolute()` renvoie le chemin absolu du fichier qui contient le code python. Nous allons faire un petit test en créant un fichier python qui contient le code ci dessous et l'enregistrer sur le bureau :

Exemple. `absolute path`

```
1 import os
2 from pathlib import Path
3 myDirectory = "."
4 p = Path(myDirectory)
5 print(p.absolute())
6 # Affiche : "C:\Users\acer\Desktop"
```

5.1.4.4 Transformer un chemin en adresse uri avec la méthode `as_uri()`

La méthode `as_uri()` est utilisée pour transformer un chemin en uri (uniforme ressource identifier)

Exemple. méthode `as_uri()`

```
1 from pathlib import Path
2 myDirectory = "C:/Users/Public/Videos/Sample Videos/
  Wildlife.wmv"
3 p = Path(myDirectory)
4 print(p.as_uri())
5 # Affiche : file:///C:/Users/Public/Videos/Sample%20Videos
  /Wildlife.wmv
```

5.1.4.5 Obtenir le chemin du dossier parent avec la méthode `parent`

La méthode `parent` permet de renvoyer le dossier parent d'un dossier existant :

Exemple. dossier parent

```

1 from pathlib import Path
2 from pathlib import Path
3 myDirectory = "C:/Users/Public/"
4 = Path(myDirectory)
5 print(p.parent) # Affiche 'C:\Users'
6 # parent renvoie aussi le dossier parent d'un fichier
7 myDirectory = "C:/Users/Public/Videos/Sample Videos/
  Wildlife.wmv"
8 p = Path(myDirectory)
9 print(p.parent) # Affiche 'C:\Users\Public\Videos\Sample
  Videos'
```

5.1.4.6 Récupération du contenu d'un dossier avec la méthode `scandir()` appliquée à la méthode `Path()`

En appliquant la méthode `scandir()` à la méthode `Path()`, on peut obtenir le contenu d'un dossier :

Exemple. récupération du contenu du répertoire 'C:/Users'

```

1 from pathlib import Path
2 import os
3 myDirectory="c:/users"
4 p = Path(myDirectory)
5 for x in os.scandir(p):
6     print(x)
```

Ce qui affiche à l'exécution :

```

<DirEntry 'acer'>
<DirEntry 'All Users'>
<DirEntry 'Default'>
<DirEntry 'Default User'>
<DirEntry 'desktop.ini'>
<DirEntry 'Public'>
```

5.1.4.7 Afficher tous les fichiers d'une extension spécifique via la méthode `glob()`

La méthode `glob()` est l'une des méthodes de l'objet `Path` permettant d'afficher la liste des fichiers d'une extension donnée :

Exemple. affichage des bibliothèques `.dll` du répertoire 'C:/Windows',

```
1 from pathlib import Path
2 p = Path('C:/Windows/')
3 for f in list(p.glob('**/*.dll')) :
4     print(f)
```

Remarque 12. Pour plus de détails sur le module Path voir la documentation officielle : <https://docs.python.org/3.0/library/os.path.html>

5.2 Mode d'ouverture d'un fichier

En langage Python, il n'est pas nécessaire d'importer une bibliothèque pour lire et écrire sur des fichiers. Il s'agit d'opérations gérées nativement dans par le langage. La première chose à faire est d'utiliser la fonction `open()` intégrée de Python pour obtenir un objet fichier (Python file object). La fonction `open()` ouvre un fichier d'une façon assez simple ! Lorsque vous utilisez la fonction `open()`, elle renvoie un objet du type file object. Les objets file object, contiennent des méthodes et des attributs pouvant être utilisés pour collecter des informations sur le fichier que vous avez ouvert. Ils peuvent également être utilisés pour manipuler le dit fichier.

Un objet fichier crée par la méthode **`open()`**, est doté de certaines propriétés permettant de lire et écrire dans un fichier. Sa syntaxe est :

```
1 f = open([nom du fichier], [mode ouverture])
```

Le `[nom du fichier]` est le nom du fichier qu'on souhaite ouvrir ou créer. Le mode d'ouverture comprend les paramètres suivants :

- **Le mode 'r'** : ouverture d'un fichier existant en lecture seule,
- **Le mode 'w'** : ouverture en écriture seule, écrasé s'il existe déjà et crée s'il n'existe pas,
- **Le mode 'a'** : ouverture et écriture en fin du fichier avec conservation du contenu existant
- **Le mode '+'** : ouverture en lecture et écriture
- **Le mode 'b'** : ouverture en mode binaire

5.3 Ouverture et lecture d'un fichier

Pour lire un fichier existant, plusieurs méthode sont disponible :

5.3.1 Lecture totale avec la méthode `read()`

La méthode **`read()`** permet de lire le contenu total ou partiel d'un fichier, après être ouvert avec la méthode **`open()`**.

La syntaxe est :

```
1 fichier.read()
```

Exemple. ouverture et lecture d'un fichier existant

```
1 f = open("myFile.txt", 'r')
2 contenu = f.read() # lecture du contenu
3 print(contenu) # impression du contenu
4 f.close() # fermeture du fichier
```

5.3.2 Lecture partielle avec la méthode read()

La méthode **read()** peut être également utilisée pour lire une partie du fichier seulement en indiquant le nombre de caractère à lire entre parenthèses :

Exemple. lecture partielle

```
1 f = open("myFile.txt", 'r')
2 contenu = f.read(20) # lecture de 20 caractère du contenu
                     # du fichier
3 print(contenu) # impression du contenu
4 f.close() # fermeture du fichier
```

Remarque 13. Après exécution de la fonction **read(n)** (n = nombre de caractères à lire), le curseur se trouve à la position **n+1**, et donc si on exécute la fonction une $2^{\text{ème}}$ fois, la lecture débutera depuis le $(n+1)^{\text{ème}}$ caractère.

5.3.3 Lecture séquentielle caractère par caractère

La méthode **read** pourra être utilisé aussi pour effectuer une lecture séquentielle caractère par caractère en utilisant la boucle **for** :

```
1 for c in fichier.read():
```

Exemple. lecture séquentielle

```
1 f = open("myFile.txt", 'r')
2 s=""
3 for c in f.read():
4     s = s + c
5 print(s)
```

La même opération peut être réalisée en utilisant la boucle **while** :

Exemple. lecture d'un fichier avec la boucle while

```
1 f = open("myFile.txt", 'r')
2 s=""
3 while 1:
4     c = f.read(1)
5     if c == " ":
6         break
7     s = s + c
8 print(s)
```

5.3.4 Lecture ligne par ligne avec les méthodes `readline()` et `readlines()`

5.3.4.1 La méthode `readline()`

La méthode `readline()` permet de lire un fichier ligne par ligne. Cette méthode pointe sur la première ligne lors de sa première exécution, ensuite sur la deuxième ligne lors de seconde exécution et ainsi à la $n^{\text{ème}}$ exécution, elle pointe vers la $n^{\text{ème}}$ ligne.

Exemple. lecture du fichier ligne par ligne

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 print(f.readline()) # affiche la ligne n°1
4 print(f.readline()) # affiche la ligne n°2
```

En combinant la méthode `readline()` avec la méthode `while()`, on peut lire la totalité des lignes d'un fichier :

Exemple. lecture de toutes les lignes avec `readline()`

```
1 f = open("myFile.txt", 'r')
2 s=""
3 while 1:
4     ligne = f.readline()
5     if(ligne == " "):
6         break
7     s = s + ligne
8 print(s) # impression de la totalité des lignes
```


5.3.4.2 La méthode `readlines()`

La méthode `readlines()`, renvoie une liste dont les éléments sont les lignes du fichier

Exemple. lecture des lignes du fichier avec `readlines()`

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 content = f.readlines()
4 print(content[0]) # impression de la première ligne
5 print(content[1]) # impression de la deuxième ligne
```

On peut aussi lire la totalité des lignes du fichier en appliquant la boucle `for` :

Exemple. lecture des lignes à l'aide de la boucle `for`

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 content = f.readlines()
4 for ligne in content :
5     print(ligne)
```

On peut donc via `readlines()`, récupérer le nombre de lignes d'un fichier en appliquant la méthode `len()` :

Exemple. nombre de lignes d'un fichier

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 content = f.readlines()
4 nombre_lignes = len(content) # récupération du nombre des
                               lignes du fichier
```

En récupérant le nombre de lignes d'un fichier, on peut donc lire la totalité de ses lignes en utilisant la boucle `for` :

Exemple. lecture de la totalité des lignes avec la boucle `for`

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 content = f.readlines()
4 n = len(content)
5 for i in range(0, n-1) :
6     print(content[i])
```

5.3.4.3 Lecture d'un fichier à une position précise avec la méthode `readlines()`

La méthode `readlines()` nous permet aussi de lire un fichier à une position bien précise :

Exemple. lecture d'un fichier depuis le caractère 10 jusqu'au caractère 20 de la troisième ligne

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 content = f.readlines()[2] #récupération de la deuxième
    ligne
4 result = content[9:19] # extraction depuis le caractère
    qui se trouve à la position 10 jusqu'à 20
5 print(result)
```

5.4 Lecture et écriture à une position donnée à l'aide de la méthode `seek()`

La méthode `seek()` permet de sélectionner une position précise pour lecture ou écriture

Exemple. lire le fichier à partir de la 6^{ème} position

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r')
3 f.seek(5) # sélection de la position 5
4 print(f.read()) #lire le fichier à partir de la 6 ème
    position
```

Exemple. écrire à partir de la 6^{ème} position

```
1 # -*- coding : utf-8 -*-
2 f = open("myFile.txt", 'r+')
3 f.seek(5) # sélection de la position 5
4 print(f.write("...")) #mettre des points sur le fichier à
    partir de la 6 ème position
```

5.5 Ouverture en mode écriture des fichiers en Python

5.5.1 Ouverture et écriture dans un fichier existant

Pour écrire dans un fichier existant, vous devez ajouter l'un des paramètres à la fonction `open()` :

1. **"a" - Append** - sera ajouté à la fin du fichier
2. **"w" - Write** - écrasera tout contenu existant
3. **"r+" Lecture et écriture** sans écraser le contenu existant

On dira alors que le fichier est ouvert en **mode écriture (write mode)**. Pour **écrire** dans fichier ouvert en **mode écriture**, on utilise la fonction `write()`.

La syntaxe est :

```
1 file.write(contenu)
```

Exemple. ouvrir un fichier et y ajouter du contenu :

```
1 # ouverture avec conservation du contenu existant
2 f = open("myFile.txt", "a")
3 f.write("Voici un contenu qui va s'ajouter au fichier
   sans écraser le contenu!")
4 f.close()
5 # ouvrir et lire le fichier après l'ajout :
6 f = open("myFile.txt", "r")
7 print(f.read())
```

Exemple. ouvrir le fichier "myFile.txt" et écrasez le contenu :

```
1 # -*- coding : utf-8 -*-
2 # ouverture avec écrasement du contenu existant
3 f = open("myFile.txt", "w")
4 f.write("Désolé ! J'ai supprimé le contenu!")
5 f.close()
6 # ouvrir et lire le fichier après l'ajout :
7 f = open("myFile.txt", "r")
8 print(f.read())
```

Remarque 14. la méthode **"w"** écrase entièrement le contenu existant du fichier.

5.5.2 Création des fichiers

Pour créer un nouveau fichier en Python, on utilise la méthode `open()`, avec l'un des paramètres suivants :

1. **"x"** - ce mode d'ouverture, crée un fichier s'il n'existe pas et renvoie une erreur si le fichier existe
2. **"a"** - **Append** - créera un fichier si le fichier spécifié n'existe pas
3. **"w"** - **Write** - créera un fichier si le fichier spécifié n'existe pas et si le fichier existe, il sera écrasé
4. **"r+"** - ouverture en mode lecture et écriture. Si le fichier n'existe pas, une erreur est renvoyée.

Exemple. Création d'un fichier nommé "myFile.txt" :

```
1 f = open ("myFile.txt", "x")
```

Résultat : un nouveau fichier vide est créé!

Exemple. Création d'un nouveau fichier s'il n'existe pas :

```
1 f = open ("myFile.txt", "w")
```

5.5.3 Ajouter des lignes à un fichier en Python avec la méthode `writelines()`

La méthode `writelines()`, permet d'ajouter une liste de chaînes ou une liste de lignes.

Exemple. ajouter une liste des lignes à un fichier

```
1 f = open ("myFile.txt", "r+")
2 l = ["ligne1\n", "ligne2\n", "ligne3\n"]
3 f.writelines(l)
4 print(f.read())
5 f.close()
```

Ce qui affiche après exécution :

ligne1

ligne2

ligne3

5.6 Récapitulatif des méthodes Python associées à un objet fichier avec description :

1. **file.close()** : ferme un fichier ouvert.
2. **file.fileno()** : retourne un descripteur entier d'un fichier.
3. **file.flush()** : vide le tampon interne.
4. **file.isatty()** : renvoie true si le fichier est connecté à un périphérique tty.
5. **file.next()** : retourne la ligne suivante du fichier.
6. **fichier.read(n)** : lit les n premiers caractères du fichier.
7. **file.readline()** : lit une seule ligne dans une chaîne ou un fichier.
8. **file.readlines()** : lit et renvoie la liste de toutes les lignes du fichier.
9. **file.seek()** : définit la position actuelle du fichier.
10. **file.seekable()** : vérifie si le fichier prend en charge l'accès aléatoire. Renvoie true si oui.
11. **file.tell()** : retourne la position actuelle dans le fichier.
12. **file.truncate(n)** : tronque la taille du fichier. Si n est fourni, le fichier est tronqué à n octets, sinon tronqué à l'emplacement actuel.
13. **file.write(str)** : écrit la chaîne **str** dans le fichier.
14. **file.writelines(séquence)** : écrit une séquence de lignes ou de chaînes dans le fichier.

5.7 Manipulation des fichiers de configuration en Python

5.7.1 Le module configparser

Le langage Python est doté d'un module nommé **configparser** qui permet d'utiliser et manipuler les fichiers de configuration similaires aux fichiers Windows du type **.ini**

le module **configparser** pourra être utilisé pour gérer les fichiers de configurations modifiables par l'utilisateur au sein d'une application. Le contenu d'un fichier de configuration peut être organisé en **sections** contenant chacune des paramètres avec des valeurs associées. Plusieurs types de valeurs d'options sont pris en charge, y compris les entiers, les valeurs à virgule flottante et les booléens.

5.7.2 Format du fichier de configuration

Le format de fichier utilisé par **configparser** est similaire à celui utilisé par les anciennes versions de Microsoft Windows. Il se compose d'une ou de plusieurs sections nommées, chacune pouvant contenir des options avec des noms et des valeurs.

1. **Les sections** de fichier de configuration sont identifiées en recherchant les lignes commençant par `[` et se terminant par `]`. La valeur entre les crochets désigne le nom de la section et peut contenir tous les caractères sauf les crochets.
2. **Les options** sont répertoriées une par ligne dans une section. La ligne commence par le nom de l'option, qui est séparée de la valeur par un signe deux-points `:` ou un signe égal `=`.
3. **Les lignes** commençant par un point-virgule `;` ou une dièse `#` sont traitées comme des commentaires et ne sont pas visibles lors de l'accès par programme au contenu du fichier de configuration.

L'exemple suivant du fichier de configuration, comporte une section nommée **settings** avec trois options, **host**, **username** et **password** :

Exemple. **configuration.ini**

```
1 [ settings ]
2 # paramètres de configuration du serveur
3 # paramètres de configuration du serveur
4 [ settings ]
5 host : 127.0.0.1
6 username : root
7 password : root
```

5.7.3 Lecture d'un fichier de configuration .ini en Python

Avec le langage Python on peut se servir du module **configparser** et la méthode **read()** de l'objet **ConfigParser** pour lire le fichier de configuration :

Exemple. Lecture du fichier **configuration.ini** en Python

```
1 from configparser import ConfigParser
2 parser = ConfigParser()
3 parser.read('configuration.ini')
4
5 # Affichage de la valeur du paramètre 'host'
6 print(parser.get('settings', 'host'))
```

5.7.4 Mettre à jour un paramètre

Pour mettre à jour un paramètre du fichier de configuration, on doit préalablement l'ouvrir en mode lecture et écriture et utiliser ensuite la méthode `set()` de la classe `ConfigParser`

Exemple. mettre à jour la valeur du paramètre 'host'

```
1  # -*- coding : utf-8 -*-
2  from configparser import ConfigParser
3  parser = ConfigParser()
4  parser.read('configuration.ini')
5
6  # Ouverture du fichier de configuration en mode lecture et
   écriture
7  file = open('configuration.ini', 'r+')
8
9  # mettre à jours la valeur du paramètre 'host'
10 parser.set('settings', 'host', 'localhost')
11 parser.write(file)
12 file.close()
```

Maintenant si vous ouvrez le fichier `configuration.ini`, vous allez constater que le paramètre `host = '127.0.0.1'` est devenu `host = 'localhost'`

5.7.5 Lecture des sections d'un fichier de configuration

La méthode `sections()` de l'objet `parser`, permet de récupérer les sections du fichier de configuration sous forme d'une liste :

Exemple. `configuration.ini`

```
1  [settings]
2  # paramètres de configuration du serveur
3  host      : 127.0.0.1
4  username  : root
5  password  : root
6
7  [Safe_Mode]
8  # http://php.net/safe-mode
9  safe_mode = Off
10 safe_mode_gid = on
11
12 [File_Uploads]
13 # http://php.net/file-uploads
14 file_uploads = On
15 upload_tmp_dir = "c:/wamp/tmp"
16 upload_max_filesize = 2M
```

Exemple. lecture des sections du fichier `configuration.ini`

```

1 from configparser import ConfigParser
2 parser = ConfigParser()
3 parser.read('configuration.ini')
4 sec = parser.sections()
5 print(sec) # Affiche : ['settings', 'Safe_Mode', '
                File_Uploads']

```

5.7.6 Lire la liste des options avec la méthode `options()`

La méthode `options()` de l'objet `ConfigParser`, permet de récupérer la liste des options d'une section du fichier de configurations :

Exemple. Liste des options de la première section :

```

1 # -*- coding : utf-8 -*-
2 from configparser import ConfigParser
3 parser = ConfigParser()
4 parser.read('configuration.ini')
5 # pointer vers la première section
6 sec = parser.sections()[0]
7 # obtenir la liste des options
8 print(sec, parser.options(sec)) # Affiche : settings ['host
                ', 'username', 'password']

```

Et en utilisant la boucle `for`, on peut obtenir toutes les sections avec leurs options :

Exemple. Liste de toutes les sections avec leurs options :

```

1 # -*- coding : utf-8 -*-
2 from configparser import ConfigParser
3 parser = ConfigParser()
4 parser.read('configuration.ini')
5
6 # Parcourt de toutes les sections
7 for sec in parser.sections():
8     # parcourt des options des différentes sections
9     print(sec, " : ", parser.options(sec))

```

Ce qui affiche à l'exécution :

```

1 settings : ['host', 'username', 'password']
2 Safe_Mode : ['safe_mode', 'safe_mode_gid']
3 File_Uploads : ['file_uploads', 'upload_tmp_dir', '
                upload_max_filesize']

```


5.7.7 Lecture totale avec la méthode items()

La méthode items(), permet de récupérer les noms des paramètres à avec leurs valeurs :

Exemple. récupération des paramètres et valeurs de la première section

```

1  # -*- coding : utf-8 -*-
2  from configparser import ConfigParser
3  parser = ConfigParser()
4  parser.read('configuration.ini')
5
6  # Pointer vers la première section
7  sec = parser.sections()[0]
8  print(sec, " : ")
9  for name, value in parser.items(sec) :
10     print(' {} = {}'.format(name, value))

```

Ce qui affiche à l'exécution :

```

1  settings :
2  host = '127.0.0.1'
3  username = 'root'
4  password = 'root'

```

Affichage complet des sections avec leurs paramètres et valeurs :

```

1  # -*- coding : utf-8 -*-
2  from configparser import ConfigParser
3  parser = ConfigParser()
4  parser.read('configuration.ini')
5
6  # Parcourt des sections
7  for sec in parser.sections() :
8     print(sec, " : ")
9     # parcourt des paramètres et valeurs
10    for name, value in parser.items(sec) :
11        print(' {} = {}'.format(name, value))

```

Affichage après exécution :

```

1  settings :
2     host = '127.0.0.1'
3     username = 'root'
4     password = 'root'
5
6  Safe_Mode :
7     safe_mode = Off
8     safe_mode_gid = on
9
10 File_Uploads :
11    file_uploads = On

```

```
12 upload_tmp_dir = "c:/wamp/tmp"
13 upload_max_filesize = 2M
```

5.7.8 Ajouter une section avec la méthode `add_section()`

Avec la méthode `add_section()` de la classe **ConfigParser**, on peut ajouter autant de sections qu'on souhaite, pour cela on doit préalablement ouvrir le fichier de configuration en mode lecture et écriture et utiliser ensuite la méthode `set()` de la classe **ConfigParser** afin de pouvoir définir et ajouter de **nouvelles options** à la section.

Exemple. Ajout d'une section nommée **mysqld**

```
1 # -*- coding: utf-8 -*-
2 from configparser import ConfigParser
3 parser = ConfigParser()
4 parser.read('configuration.ini')
5
6 # Ouverture du fichier de configuration
7 file = open('configuration.ini', 'r+')
8
9 # Ajout d'une nouvelle section mysqld
10 parser.add_section('mysqld')
11 # Définition et ajout de nouvelles options
12 parser.set('mysqld', 'port', '3306')
13 parser.set('mysqld', 'table_cache', '64')
14 parser.write(file)
15 file.close()
```

Chapitre 6

Python et les bases de données SQLite

6.1 A propos des bases de données SQLite3

SQLite est un moteur de base de données écrit en langage C. Il s'agit d'un logiciel qui permet aux utilisateurs d'interagir avec une **base de données relationnelle**. Dans SQLite, une base de données est stockée dans un **seul fichier** ce qui la distingue des autres moteurs de base de données. Ce fait permet une grande accessibilité : copier une base de données n'est pas plus compliqué que copier le fichier qui stocke les données, partager une base de données peut signifier envoyer une pièce jointe à un e-mail.

6.2 Création de tables et de bases de données SQLite3

SQLite est une bibliothèque qui fournit une base de données légère sur disque ne nécessitant pas de processus serveur distinct et permet d'accéder à la base de données à l'aide d'une variante du langage de requête SQL. Certaines applications peuvent utiliser SQLite pour le stockage de données interne. Il est également possible de prototyper une application utilisant SQLite, puis de transférer le code dans une base de données plus grande telle que PostgreSQL ou Oracle.

Pour utiliser le module, vous devez d'abord créer un objet **Connection** qui représente la base de données. Dans l'exemple ci-dessous, les données seront stockées dans le fichier **mabase.db** :

```

1 import sqlite3
2 conn = sqlite3.connect('mabase.db')

```

Remarque. importante! Vous n'êtes pas obligé de créer la base de données **mabase.db**, mais elle sera créée automatiquement dans le même répertoire que le fichier Python!

Une fois que vous avez une connexion, vous pouvez créer un objet **Cursor** et appeler sa méthode **execute()** pour exécuter des commandes **SQL** :

```

1 # Créer un cursor
2 cur = conn.cursor()

```

Et maintenant si on veut créer une table au sein de la base **SQLite3 mabase.db**, il suffit d'utiliser la commande **CREATE TABLE nom_de_la_table :**

Code complet :

```

1 # -*- coding : utf-8 -*-
2 import sqlite3
3 conn = sqlite3.connect('mabase.db')
4 # Créer un cursor
5 cur = conn.cursor()
6 # Création de la requete
7 req = "CREATE TABLE students(id INTEGER PRIMARY KEY
      AUTOINCREMENT, name TEXT NOT NULL, email TEXT NOT NULL
      )"
8 # Exécution de la requete
9 cur.execute(req)
10 # Envoyer la requete
11 conn.commit()
12 # Fermer la connexion
13 conn.close

```

6.3 Insertion de données

L'insertion de données en environnement SQLite3 est exactement identique au cas du MySQL :

```

1 # Insérer une ligne de données
2 cur.execute("INSERT INTO students ('nom', 'email') VALUES
      ('Albert', 'albert@gmail.com')")
3 # Commettre ou engager les données
4 conn.commit()

```

Code complet :

```

1  # -*- coding : utf-8 -*-
2  import sqlite3
3  conn = sqlite3.connect('mabase.db')
4  cur = conn.cursor()
5  # Insérer une ligne de données
6  cur.execute("INSERT INTO students('nom','email') VALUES ('
      Albert', 'albert@gmail.com')")
7  # Engager l'opération
8  conn.commit()
9  # Fermer la connexion
10 conn.close()

```

6.4 Insertion des données de variables dans une table SQLite

Quand on a inséré directement les données au sein de la requête comme on a fait dans l'exemple ci-dessus, aucun problème n'a été rencontré !

```

1  cur.execute("INSERT INTO students('nom','email') VALUES ('
      Albert', 'albert@gmail.com')")

```

Imaginez que les données qu'on souhaite insérer, sont des valeurs de variables récupérées depuis un autre fichier ou provenant d'un formulaire d'enregistrement... Dans ce cas l'insertion des données de cette façon est totalement erronée !:

```

1  nom = "Albert"
2  email = "albert@gmail.com"
3  cur.execute("INSERT INTO students('nom','email') VALUES (
      nom, email)")
4  # TOTALEMENT FAUX !

```

ATTENTION! TOTALEMENT FAUX! Puisque les variables **nom** et **email** ne seront pas interprétées !

Pour corriger l'erreur, on utilise la méthode de **formatage** des chaînes à l'aide du **symbole : " ? "**

```

1  nom = 'Albert'
2  email = 'albert@gmail.com'
3  age = 22
4  cur = conn.cursor()
5  cur.execute("Insert into students (nom,email,age) values
      (?, ?, ?)",(nom, email, age))

```

Code complet :

```

1  # -*- coding : utf-8 -*-
2  import sqlite3
3  conn = sqlite3.connect('mabase.db')
4  nom = 'Albert'
5  email = 'albert@gmail.com'
6  age = 22
7  # Créer un cursor
8  cur = conn.cursor()
9  cur.execute("Insert into students (nom,email,age) values
              (?, ?, ?)", (nom, email, age))
10 conn.commit()
11 conn.close()

```

6.5 Affichage des données d'une table SQLite3

Maintenant, il est tout à fait légitime de se demander si tout a été bien réglé : **création de la table students** au sein de la base de données SQLite3 , **insertion de données** au sein de la table **students**...

6.5.0.1 Création d'un cursor pour exécuter une requête de sélection

```

1  cur = conn.cursor()
2  result = cur.execute("select*from students ")

```

6.5.0.2 Parcourir les résultats de la sélection

Pour afficher les données, on va parcourir l'objet **cursor** par un compteur **row**. La variable **row** qui fait le parcourt est un **objet tuple** dont les constituants sont les valeurs des champs : **id**, **nom**, **email**, **age**...

```

1  for row in result :
2      print("ID : ",row[0])
3      print("nom : ",row[1])
4      print("Email : ",row[2])
5      print("Age : ",row[3])

```

6.6 Mise à jour des données SQLite3

On souhaite parfois mettre à jours les données de certain champs pour un certain id déterminé. Pour ce faire, on utilise l'instruction UPDATE :

Exemple. mise à jour des champs : **nom** et **email** pour un identifiant **id** sélectionné **idSelect**

```
1 new_nom = 'David'
2 new_email = 'albert@gmail.com'
3 cur = conn.cursor()
4 cur.execute("UPDATE students set 'nom' = ? , 'email' = ?
              WHERE 'id' = ?" , (new_nom, new_email, idSelect))
5 conn.commit()
6 conn.close()
```

Remarque 15. On peut aussi utiliser un objet dictionnaire dont les valeurs sont les nouvelles valeurs des champs

Exemple. .

```
1 new_nom = 'David'
2 new_email = 'albert@gmail.com'
3 cur = conn.cursor()
4 cur.execute("UPDATE students set 'nom' = :new_nom , 'email'
              = :new_email WHERE 'id' = :idSelect " , {"new_nom" :
              new_nom, "new_email" : new_email, "idSelect" : idSelect})
5 conn.commit()
6 conn.close()
```

6.7 Récupération totale des données sous forme d'un tableau à deux dimension

Pour récupérer la totalité des enregistrements de la table sqlite, on utilise la méthode **fetchall** :

Exemple. .

```
1 cur = conn.cursor()
2 cur.execute("select * from students")
3 rows = cur.fetchall()
4 print(rows) # affiche le contenu en entier
5 print(rows[0] # affiche la 1ere ligne
```

6.8 Exportation du contenu de la base SQLite vers une base sql (SQLite3 dump)

SQLite3 offre la possibilité d'exporter le contenu d'une base de données SQLite3 vers un fichier .sql via la méthode **iterdump()** :

```

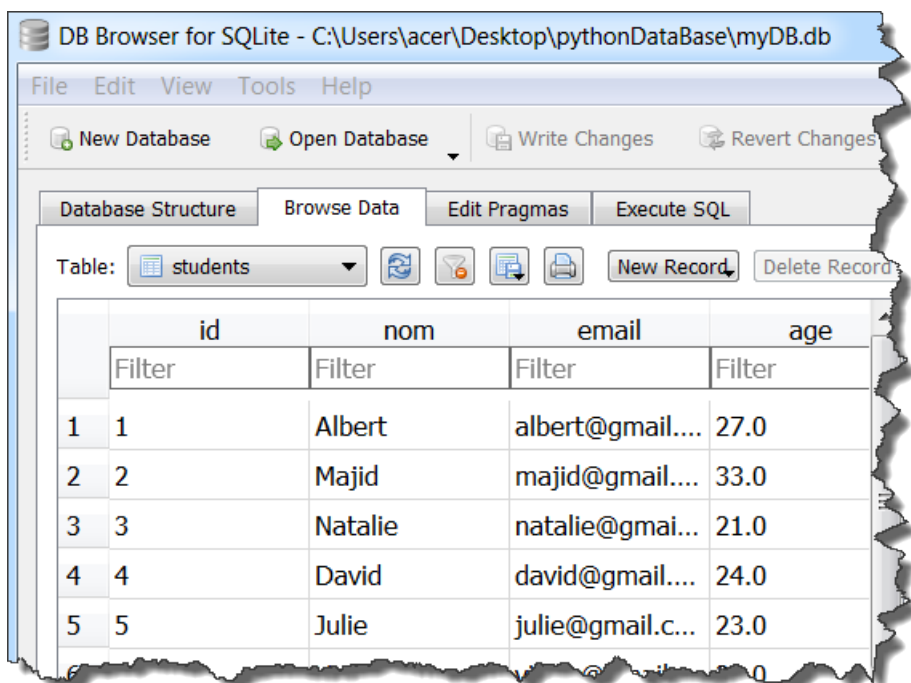
1 # Convert file database.db to SQL dump file dump.sql
2 import sqlite3
3
4 conn = sqlite3.connect('database.db')
5 with open('dump.sql', 'w') as file :
6     for line in conn.iterdump():
7         file.write('%s\n' % line)
8 conn.close()

```

6.9 Éditeur WYSIWYG SQLite3

Tout a été fait en noir ! Jusqu'à présent vous n'avez encore rien vu, ni table ni données... Pour répondre à cette question, je vous informe qu'il y a de nombreux utilitaires permettant d'explorer les bases de données SQLite3. Je vous recommande **DB Browser for SQLite** qui est gratuit et très simple d'usage :

1. Téléchargez [DB Browser for SQLite](#),
2. Installez le,
3. Lancez DB Browser
4. Depuis le menu **File** → cliquez sur le sous menu **Open Database**
5. Sélectionnez ensuite votre base de donnée **mabase.db**
6. Cliquez finalement sur Browse data pour voir votre table students avec les données que vous venez d'insérer :



Deuxième partie

Interfaces Graphiques

Chapitre 7

Les bibliothèques d'interfaces graphiques

Les interfaces graphiques en Python

Python fournit diverses options pour développer des interfaces **graphiques GUI** :

1. **Tkinter** : Tkinter est l'interface Python de la bibliothèque GUI Tk livrée avec Python. Nous allons l'étudier en détail sur ce chapitre.
2. **wxPython** : Ceci est une implémentation en Python libre et open source Python de l'interface de programmation wxWidgets.
3. **PyQt** : Il s'agit également d'une interface Python pour une bibliothèque d'interface graphique Qt populaire multiplate-forme.
4. **JPython** : JPython est un outil Python pour Java, qui donne aux scripts Python un accès transparent aux bibliothèques de classes Java.

Il existe de nombreuses autres interfaces graphiques disponibles, que vous pouvez trouver sur le net.

Remarque. Dans le présent ouvrage, nous allons nous **focaliser** autour de la **bibliothèque Tkinter**

7.1 Première fenêtre graphique avec Tkinter

Tkinter est la bibliothèque d'**interface graphique standard** pour **Python**. Python, lorsqu'il est combiné à **Tkinter**, fournit un moyen ra-

pide et facile pour créer des applications graphiques. **Tkinter** fournit une puissante interface orientée objet simple et conviviale.

La création d'une application graphique à l'aide de Tkinter est une tâche assez facile. Tout ce que vous avez à faire est de suivre les étapes suivantes :

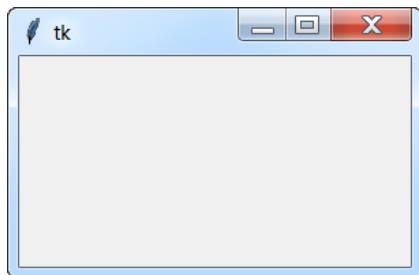
1. *Importez le module Tkinter.*
2. *Créez la fenêtre principale de l'application graphique via une instantiation sur la classe Tk.*
3. *Entrez la boucle d'évènements principale pour agir contre chaque évènement déclenché par l'utilisateur.*

Exemple. création d'une simple fenetre Tkinter

```

1  # -*- coding : utf-8 -*-
2  # 1)- Importation de la bibliothèque tkinter
3  from tkinter import *
4
5  # 2)- Création d'une fenêtre tkinter en faisant une
    instantiation de la classe Tk
6  maFenetre = Tk()
7
8  # vos widgets ici : bouton de commande, champ de saisie ,
    labels...
9
10 # 3)- Entrez la boucle d'événements principale
11 maFenetre.mainloop()
```

Ce qui affiche après exécution :



7.2 Les widgets Tkinter

La bibliothèque Tkinter fournit divers contrôles, tels que des boutons, des étiquettes et des zones de texte utilisées dans une application graphique. Ces contrôles sont communément appelés **widgets**.

Il existe actuellement **15 types de widgets dans Tkinter**. Nous présentons ici les noms de ces widgets ainsi qu'une brève description :

1. **Button** : le widget Button permet de créer des boutons pour votre application.
2. **Canva** : le widget Canva permet de dessiner des formes, telles que des lignes, des ovales, des polygones et des rectangles, dans votre application.
3. **Checkbutton** : le widget Checkbutton permet d'afficher un certain nombre d'options sous forme de cases à cocher. L'utilisateur peut sélectionner plusieurs options à la fois.
4. **Entry** : le widget Entry est utilisé pour afficher un champ de texte d'une seule ligne permettant d'accepter les valeurs d'un utilisateur.
5. **Frame** : le widget Frame (cadre) est utilisé en tant que widget conteneur pour organiser d'autres widgets.
6. **Label** : le widget Label est utilisé pour fournir une légende ou description pour les autres widgets. Il peut aussi contenir des images.
7. **Listbox** : le widget Listbox est utilisé pour fournir une liste d'options à un utilisateur.
8. **menubutton** : le widget menubutton est utilisé pour afficher les menus dans votre application.
9. **Menu** : le widget Menu est utilisé pour fournir diverses commandes à un utilisateur. Ces commandes sont contenues dans Menubutton.
10. **Message** : le widget Message est utilisé pour afficher des champs de texte multilignes permettant d'accepter les valeurs d'un utilisateur.
11. **Radiobutton** : le widget Radiobutton est utilisé pour afficher un certain nombre d'options sous forme de boutons radio. L'utilisateur ne peut sélectionner qu'une option à la fois.
12. **Scale** : le widget Echelle est utilisé pour fournir un widget à curseur.
13. **Scrollbar** : le widget Scrollbar ou barre de défilement est utilisé pour ajouter une fonctionnalité de défilement à divers widgets, tels que les zones de liste.
14. **Text** : le widget Text est utilisé pour afficher du texte sur plusieurs lignes.
15. **Toplevel** : le widget Toplevel est utilisé pour fournir un conteneur de fenêtre séparé.
16. **Spinbox** : le widget Spinbox est une variante du widget standard **Tkinter Entry**, qui peut être utilisé pour sélectionner un nombre fixe de valeurs.

17. **PanedWindow** : le widget PanedWindow est un conteneur pouvant contenir un nombre quelconque de volets, disposés horizontalement ou verticalement.
18. **LabelFrame** : un labelframe est un simple widget de conteneur. Son objectif principal est d'agir comme un intercalaire ou un conteneur pour les dispositions de fenêtre complexes.
19. **tkMessageBox** : ce module est utilisé pour afficher des boîtes de message dans vos applications.

7.2.1 Le widget Button

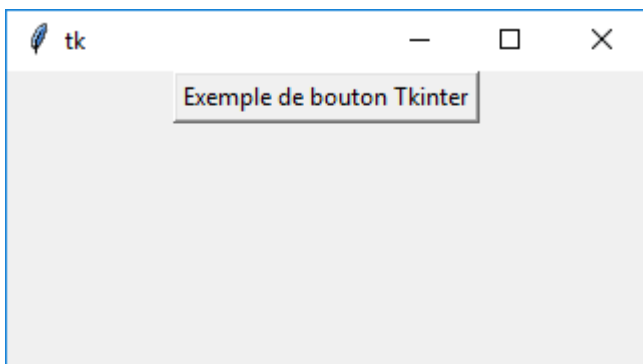
Le **widget Button** est un widget **Tkinter** standard, qui est utilisé pour différents types de boutons. Un bouton est un widget avec lequel l'utilisateur peut interagir, c'est-à-dire que si le bouton est enfoncé par un clic de souris, une action peut être lancée. Ils peuvent également contenir du texte et des images comme des étiquettes... Alors que les étiquettes peuvent afficher du texte dans différentes polices, un bouton ne peut afficher du texte que dans une seule police. Le texte d'un bouton peut s'étendre sur plusieurs lignes.

Pour créer un bouton de commande sur une fenêtre Tkinter, rien de plus simple, il suffit d'utiliser

```
1 Nom_du_bouton = Button( Nom_de_la_fenêtre, text ="Texte
    du bouton", option = Value , ... )
```

Exemple.

```
1
2 # coding : utf-8
3 from tkinter import*
4 maFenetre = Tk()
5
6 #Création d'un bouton de commande
7 b = Button(maFenetre , text = "Exemple de bouton Tkinter")
8
9 # Placer le bouton sur la fenêtre
10 b.pack()
11
12 maFenetre.mainloop()
```



Associer une action à un bouton de commande

Généralement un bouton de commande dans une interface graphique quelconque est utilisé pour déclencher une action quand on clique sur ce dernier comme fermer la fenêtre, afficher une boîte de dialogue, afficher un texte sur un label... Nous commençons par un exemple très simple : un bouton qui permet de fermer la fenêtre.

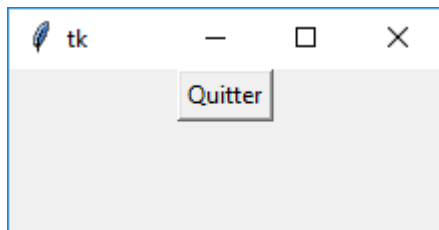
Exemple. bouton quitter l'application

```

1  # -*- coding : utf-8 -*-
2  from tkinter import *
3  maFenetre = Tk()
4
5  #Création d'un bouton de commande
6  b = Button(maFenetre , text = "Quitter" , command = quit )
7
8  # Placer le bouton sur la fenêtre
9  b.pack()
10
11 maFenetre.mainloop()

```

Ce qui affiche à l'exécution :



Avec l'action : "quitter la fenêtre en cliquant sur le bouton"

Remarque. On peut aussi créer une **méthode** qui réalise l'**action** du **click**

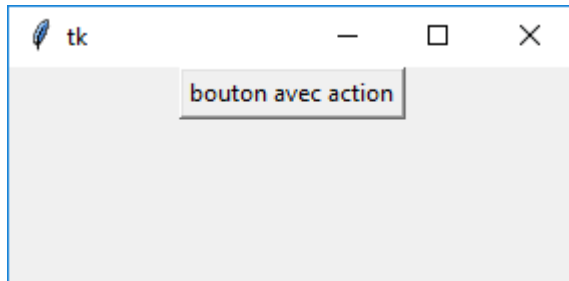
Exemple. (méthode associée à un bouton de commande)

```

1 # coding : utf-8
2 from tkinter import *
3
4 def b_action() :
5     print("Vous avez cliqué sur le bouton !")
6
7 maFenetre = Tk()
8 maFenetre.geometry("400x200")
9
10 #Création d'un bouton de commande
11 b = Button(maFenetre , text = "bouton avec action" ,
12           command = b_action)
13
14 # Placer le bouton sur la fenêtre
15 b.pack()
16
17 maFenetre.mainloop()
18
19 # affiche au click : Vous avez cliqué sur le bouton !

```

Ce qui affiche après exécution :



Les options associées à un bouton Tkinter

1. **activebackground** : couleur de fond lorsque le bouton est sous le curseur.
2. **activeforeground** : couleur de premier plan lorsque le bouton est sous le curseur.
3. **bd** : largeur de la bordure en pixels. La valeur par défaut est 2.
4. **bg** : couleur d'arrière plan.

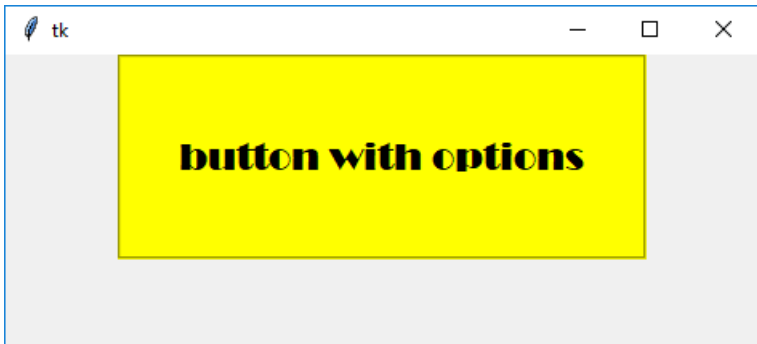
5. **command** : méthode à appeler lorsque le bouton est cliqué.
6. **fg** : couleur de (texte).
7. **font** : police de caractères à utiliser pour le libellé du bouton.
8. **height** : hauteur du bouton.
9. **highlightcolor** : la couleur de la surbrillance du focus lorsque le widget a le focus.
10. **image** : image à afficher sur le bouton (au lieu du texte).
11. **justify** : comment afficher plusieurs lignes de texte : GAUCHE pour justifier à gauche chaque ligne ; CENTRE pour les centrer ; ou RIGHT pour justifier à droite.
12. **padx** : remplissage supplémentaire à gauche et à droite du texte.
13. **pady** : rembourrage supplémentaire au-dessus et au-dessous du texte.
14. **relief** : spécifie le type de bordure. Certaines des valeurs sont SUNKEN, RAISED, GROOVE et RIDGE.
15. **state** : cette option sur DÉSACTIVÉ pour griser le bouton et le rendre insensible. A la valeur ACTIVE lorsque la souris est dessus. La valeur par défaut est NORMALE.
16. **underline** : la valeur par défaut est -1, ce qui signifie qu'aucun caractère du texte du bouton ne sera souligné. S'il n'est pas négatif, le caractère de texte correspondant sera souligné.
17. **width** : largeur du bouton.
18. **wrplength** : si cette valeur est définie sur un nombre positif, les lignes de texte seront enveloppées pour tenir dans cette longueur.

Exemple. (bouton Tkinter avec options)

```

1  # coding : utf-8
2  from tkinter import *
3
4  maFenetre = Tk()
5  maFenetre.geometry("500x200")
6
7  #Création d'un bouton de commande
8  b = Button(maFenetre , text = "bouton action" , width =
      20 , height= 4 ,\
9      relief = GROOVE , bg="Yellow" , font=('broadway
      ', 18))
10
11 # Placer le bouton sur la fenêtre
12 b.pack()
13
14 maFenetre.mainloop()
```

Ce qui affiche après exécution :



7.2.2 Le widget label

Un **widget Label** implémente une boîte d'affichage où vous pouvez placer du **texte**, des **images**... Le texte affiché par ce widget peut être mis à jour à tout moment à l'aide d'un bouton de commande ou une variable du type **StringVar()**.

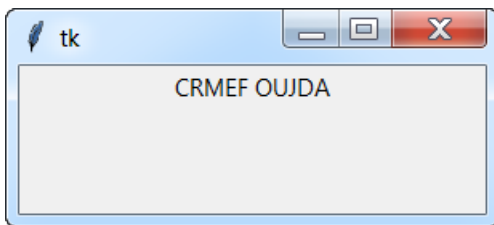
L'insertion d'un **label** sur une fenêtre **Tkinter** est semblable à celui d'un bouton de commande :

```
1 Nom_du_label = Label( Nom_de_la_fenêtre , text ="Texte du
    label ")
```

Exemple. (un simple label)

```
1 # -*- coding : utf-8 -*-
2 from tkinter import*
3 maFenetre = Tk()
4
5 #Création du label
6 lbl = Label( maFenetre , text = "CRMEF OUJDA" )
7 # Placer le label sur la fenêtre
8 lbl.pack()
9
10 maFenetre.mainloop()
```

Ce qui affiche après exécution :



Remarque 16. On peut aussi utiliser un texte variable, afin de donner la possibilité à l'utilisateur de modifier le texte :

Exemple. texte variable

```
1 # coding : utf-8
2 from tkinter import *
3 fen = Tk()
4 var = StringVar()
5 label = Label( fen , textvariable=var)
6 var.set( "CRMEF OUJDA" )
7 label.pack()
8 fen.mainloop()
```

On peut aussi changer le texte via une action associé à un bouton de commande

Exemple. ...

```
1 # coding : utf-8
2 from tkinter import *
3 fen = Tk()
4 def action() :
5     var.set( "J'ai changé le texte en cliquant" )
6 var = StringVar()
7 label = Label( fen , textvariable=var)
8 var.set( "CRMEF OUJDA" )
9 b = Button( fen , text = "Essayez" , command=action )
10 b.pack()
11 label.pack()
12 fen.mainloop()
```

Les options associées à un label Tkinter

1. **activebackground** : définit la couleur de fond du label au survole de la souris.
2. **activeforeground** : définit la couleur du label au survole de la souris.

3. **anchor** : précise la position du texte. La valeur par défaut est 'center'. Et par suite le texte d'un label est centré par défaut.
4. **bg (ou background)** : couleur du background du label.
5. **bitmap** : affiche une image sur le label
6. **bd (ou borderwidth)** : épaisseur de la bordure autour du label, la valeur par défaut est 2 pixels.
7. **compound** : cette option sert à préciser l'orientation relative de l'image par rapport au texte dans le cas où vous souhaitez afficher à la fois un texte et un graphique sur le label. Les valeur peuvent-être 'left', 'right', 'center', 'bottom' ou 'top'. Si par exemple **compound=BOTTOM**, l'image sera affiché en-dessous du texte.
8. **cursor** : forme du pointeur de la souris au survole du label.
9. **disabledforeground** : précise la couleur d'avant plan lorsque le label est désactivé.
10. **font** : précise le type et la taille du police (avec l'option text ou textvariable)
11. **fg (ou foreground)** : couleur du texte du label.
12. **height** : hauteur du label en nombre de lignes (non en pixels), le label s'ajuste par défaut à son contenu.
13. **highlightbackground** : cette option définit la couleur de mise en valeur du focus quand le widget l'a perdu.
14. **highlightcolor** : couleur du focus quand le widget l'a obtenu.
15. **highlightthickness** : définit l'épaisseur de la ligne de mise en valeur du focus.
16. **image** : cette option permet d'afficher une image dans un label.
17. **justify** : définit l'alignement du texte : 'left' pour un alignement à gauche, 'center' pour centrer et 'right' pour un alignement à droite.
18. **padx** : permet d'insérer un espace horizontal à gauche et à droite dans le label. Sa valeur par défaut est 1.
19. **pady** : option similaire à padx mais elle agit verticalement.
20. **relief** : définit l'apparence de la bordure autour du label, la valeur par défaut est 'flat'.
21. **state** : cette option est mise par défaut à l'état 'normal'. Les autres états possibles sont 'disabled' et 'active'.
22. **takefocus** : par défaut un label n'obtient pas le focus, mais si vous souhaitez que le label l'obtienne, mettez 1 pour cette option.

23. **text** : permet d'afficher une ou plusieurs ligne de texte dans un label. Pour forcer le retour à la ligne on utilise le caractère '\n'.
24. **textvariable** : permet de lier le texte du label à une variable du type StringVar afin de pouvoir changer le texte du label.
25. **underline** : par défaut, underline=-1, ce qui signifie aucun soulignement. Si on souhaite souligner un des caractères du texte on précise sa position (à partir de 0).
26. **width** : définit la largeur du label en nombre de caractères (non en pixels). Un label s'ajuste par défaut à son contenu lorsque cette option n'est pas précisée.
27. **wraplength** : si cette valeur est définie sur un nombre positif, les lignes de texte seront enveloppées pour tenir dans cette longueur.

Exemple. (Label Tkinter avec options)

```

1 from tkinter import *
2
3 root = Tk()
4 root.geometry('400x200')
5 var = StringVar()
6 label = Label( root , textvariable=var , relief=RAISED , \
7               font=('broadway' , 18) , bg='yellow' ,
               wraplength=150)
8
9 var.set("Voici un exemple de label avec options")
10 label.pack()
11 root.mainloop()

```



7.2.3 Le champ de saisie Entry

Définition et syntaxe générale du widget Entry

Le **widget Entry** en **Tkinter**, permet de créer une **zone de saisie** sur une seule ligne, pouvant servir à la récupération des données saisie par l'utilisateur afin les utiliser comme variables (formulaire d'inscription, formulaire d'identification...). La syntaxe est :

```
1 Nom_du_champ = Entry( Objet_parent , options ... )
```

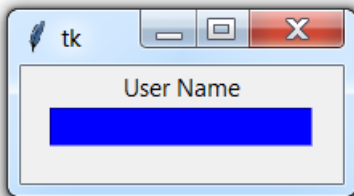
Syntaxe pour créer un widget Entry sur une fenêtre

```
1 champ_saisie = Entry (master , option , ...)
2 # master : fenêtre parente.
3 #options : la liste des options pour le widget Entry.
```

Exemple. Widget Entry

```
1 from tkinter import *
2 root = Tk()
3 user_Entry = Entry(root , bg="blue")
4 user = Label(root , text = "User Name")
5 user.pack()
6 user_Entry.pack()
7 root.mainloop()
```

Ce qui affiche après exécution :



Liste des options pour le widget Entry

1. **bg** : définit la couleur d'arrière-plan du widget Entry.
2. **bd** : définit la taille des bordures. La valeur par défaut est 2 pixels.
3. **command** : définit et associe une commande
4. **cursor** : définit le motif du curseur de la souris
5. **font** : définit la police qui sera utilisée pour le texte.

6. **exportselection** : par défaut, si vous sélectionnez du texte dans un widget Entry, il peut être exporté vers le Presse-papiers. Pour éviter cette exportation, utilisez `exportselection = 0`.
7. **fg** : définit la couleur qui sera utilisée pour le texte
8. **highlightcolor** : Il représente la couleur de surbrillance à utiliser pour le rectangle qui est dessiné autour du widget lorsqu'il a le focus.
9. **justify** : Il spécifie comment le texte est organisé s'il contient plusieurs lignes. 10relief Spécifie le type de bordure. Sa valeur par défaut est FLAT.
10. **selectbackground** : la couleur d'arrière-plan du texte sélectionné.
11. **selectborderwidth** : définit la largeur de la bordure à utiliser autour du texte sélectionné. La valeur par défaut est un pixel.
12. **selectforeground** : définit la couleur de premier plan du texte sélectionné.
13. **show** : est utilisé pour afficher ou masquer les caractères. Exemple pour un mot de passe on utilise `show = "*" .`
14. **state** : la valeur par défaut est `state = NORMAL`, mais vous pouvez utiliser `state = DISABLED` pour masquer le contrôle et le rendre inactif.
15. **textvariable** : Pour pouvoir extraire le texte actuel de votre widget Entry, vous devez définir cette option sur une instance de la classe **StringVar**.
16. **width** : définit la largeur du widget
17. **xscrollcommand** : ajoute une barre de défilement au widget.

Les méthodes associés au widget Entry

1. **delete** : (first, last = None) : Supprime les caractères du widget, en commençant par celui qui est à l'index first, jusqu'au dernier last
2. **get()** : renvoie le texte actuel de l'entrée sous forme de variable chaîne.
3. **icursor(index)** : place le curseur d'insertion juste avant le caractère à l'index donné.
4. **index(index)** : décale le contenu de l'entrée de sorte que le caractère à l'index donné soit le caractère visible le plus à gauche. N'a aucun effet si le texte est entièrement contenu dans l'entrée.

5. **insert(index, s)** : insère la chaîne s avant le caractère à l'index donné.
6. **select_adjust (index)** : cette méthode permet de s'assurer que la sélection inclut le caractère à l'index spécifié.
7. **select_clear ()** : efface la sélection. S'il n'y a pas actuellement de sélection, n'a aucun effet.
8. **select_form(index)** : définit la position de l'index d'ancrage sur le caractère spécifié par l'index.
9. **select_present()** : s'il y a une sélection, renvoie vrai, sinon renvoie faux.
10. **select_range(début, fin)** : sélectionne les caractères qui existent dans la plage spécifiée.
11. **select_to(index)** : sélectionne tous les caractères du début à l'index spécifié.
12. **xview(index)** : utilisé pour lier le widget de saisie à une barre de défilement horizontale.
13. **xview_scroll()** : utilisé pour faire défiler l'entrée horizontalement.

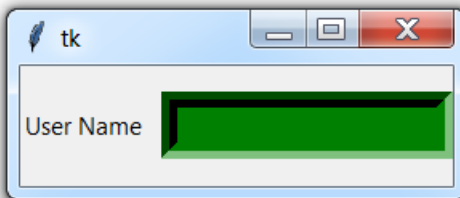
Exemple. widget Entry

```

1 from tkinter import *
2 root = Tk()
3 user = Label(root, text = "User Name")
4 user.pack( side = LEFT )
5 user_Entry = Entry(root, bd = 10, bg="green")
6 user_Entry.pack( side=RIGHT )
7 root.mainloop()

```

Ce qui affiche après exécution :



7.2.4 Associer un évènement à un Widget Tkinter

Les évènements en Tkinter seront traités plus en détails dans les paragraphes suivants, mais ici nous donnons seulement une idée d'un cas simple d'évènement associé à un champ **Entry** :

7.2.4.1 Associer un évènement à un champ Entry (bind action)

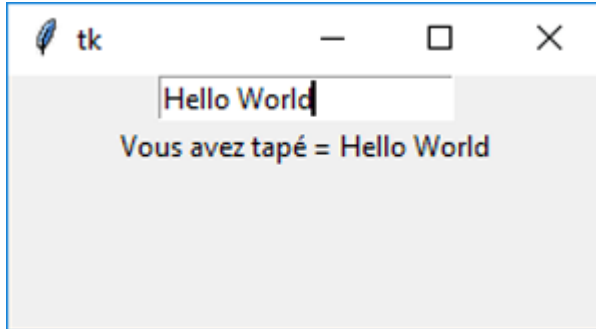
On peut associer un évènement à un champ de saisie à l'aide de la méthode **bind()** :

```
1 entry.bind("<Return>", actionEvent)
```

Exemple. Évènement associé à un champ de saisie :

```
1 from tkinter import *
2 def actionEvent(event):
3     lbl.configure(text = "Vous avez tapé = " + entry.get())
4
5 root = Tk()
6 entry = Entry(root)
7
8 # Association de l'évènement actionEvent au champ de saisie
9 entry.bind("<Return>", actionEvent)
10 lbl = Label(root, text = ".....")
11 entry.pack()
12 lbl.pack()
13 root.mainloop()
```

Et maintenant en tapant un texte "**Hello World**" à titre d'exemple et en appuyant sur la **touche entrée** du clavier on obtient :



7.2.5 Le widget Text

7.2.5.1 Syntaxe & description du widget Text

Le widget **Text** offre des fonctionnalités avancées vous permettant d'éditer un texte **multi-ligne** et de formater son affichage, en modifiant sa couleur et sa police...

Vous pouvez également utiliser des structures élégantes telles que des onglets et des marques pour localiser des sections spécifiques du texte et appliquer des modifications à ces zones. De plus, vous pouvez incorporer des fenêtres et des images dans le texte car ce **widget** a été conçu pour gérer à la fois le texte brut et le texte mis en forme.

Syntaxe du widget Text

```
1 Nom_du_widget_Text = Texte (objet_parent , options , ...)
```

Le paramètre parent : représente l'objet parent : fenêtre parent, frame...

options : représente la liste des options pour le widget.

7.2.5.2 Les options disponibles pour le widget Text

Voici la liste des options les plus utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules :

1. **bg** : la couleur d'arrière-plan par défaut du widget de texte.
2. **bd** : la largeur de la bordure autour du widget de texte. La valeur par défaut est 2 pixels.
3. **cursor** : le curseur qui apparaîtra lorsque la souris survolera le widget texte.
4. **exportselection** : normalement, le texte sélectionné dans un widget de texte est exporté pour être sélectionné dans le gestionnaire de fenêtres. Définissez `exportselection = 0` si vous ne voulez pas ce comportement.
5. **font** : la police par défaut pour le texte inséré dans le widget.
6. **fg** : la couleur utilisée pour le texte (et les bitmaps) dans le widget. Vous pouvez changer la couleur pour les régions marquées; cette option est juste la valeur par défaut.
7. **height** : la hauteur du widget en lignes (pas en pixels!), Mesurée en fonction de la taille de la police actuelle.
8. **highlightbackground** : la couleur du focus est mise en surbrillance lorsque le widget de texte n'a pas le focus.
9. **highlightcolor** : la couleur du focus est mise en surbrillance lorsque le widget de texte a le focus.
10. **highlightthickness** : l'épaisseur du focus est mise en évidence. La valeur par défaut est 1. Définissez l'épaisseur de la sélection = 0 pour supprimer l'affichage de la surbrillance.

11. **insertbackground** : la couleur du curseur d'insertion. Le défaut est noir.
12. **insertborderwidth** : taille de la bordure 3D autour du curseur d'insertion. La valeur par défaut est 0.
13. **insertofftime** : le nombre de millisecondes pendant lequel le curseur d'insertion est désactivé pendant son cycle de clignotement. Définissez cette option sur zéro pour supprimer le clignotement. La valeur par défaut est 300.
14. **insertontime** : le nombre de millisecondes pendant lequel le curseur d'insertion est activé pendant son cycle de clignotement. La valeur par défaut est 600.
15. **insertwidth** : largeur du curseur d'insertion (sa hauteur est déterminée par l'élément le plus grand de sa ligne). La valeur par défaut est 2 pixels.
16. **padx** : la taille du remplissage interne ajouté à gauche et à droite de la zone de texte. La valeur par défaut est un pixel.
17. **pady** : la taille du remplissage interne ajouté au-dessus et au-dessous de la zone de texte. La valeur par défaut est un pixel.
18. **relief** : l'apparence 3-D du widget de texte. La valeur par défaut est relief = SUNKEN.
19. **selectbackground** : la couleur de fond à utiliser pour afficher le texte sélectionné.
20. **selectborderwidth** : la largeur de la bordure à utiliser autour du texte sélectionné.
21. **spacing1** : cette option spécifie combien d'espace vertical supplémentaire est placé au-dessus de chaque ligne de texte. Si une ligne est renvoyée à la ligne, cet espace est ajouté uniquement avant la première ligne occupée à l'écran. La valeur par défaut est 0.
22. **spacing2** : cette option spécifie la quantité d'espace vertical supplémentaire à ajouter entre les lignes de texte affichées lorsqu'une ligne logique est renvoyée à la ligne. La valeur par défaut est 0.
23. **spacing3** : cette option spécifie combien d'espace vertical supplémentaire est ajouté en dessous de chaque ligne de texte. Si une ligne est renvoyée à la ligne, cet espace est ajouté uniquement après la dernière ligne occupée à l'écran. La valeur par défaut est 0.
24. **state** : normalement, les widgets de texte répondent aux événements de clavier et de souris; set state = NORMAL pour obtenir ce comportement. Si vous définissez state = DISABLED, le widget

- texte ne répondra pas et vous ne pourrez pas non plus modifier son contenu par programme.
25. **tabs** : cette option contrôle la manière dont les caractères de tabulation positionnent le texte.
 26. **width** : la largeur du widget en caractères (pas en pixels !), Mesurée en fonction de la taille de la police actuelle.
 27. **wrap** : cette option contrôle l'affichage des lignes trop larges. Définissez `wrap = WORD` et la ligne sera coupée après le dernier mot qui convient. Avec le comportement par défaut, `wrap = CHAR`, toute ligne trop longue sera brisée par n'importe quel caractère.
 28. **xscroll** : pour que le widget texte puisse défiler horizontalement, définissez cette option sur la méthode `set()` de la barre de défilement horizontale.
 29. **yscroll** : pour que le widget texte puisse défiler verticalement, définissez cette option sur la méthode `set()` de la barre de défilement verticale.

7.2.5.3 Les méthodes associées au widget Text

Voici la liste des principales **méthodes** associées à l'objet **Text**

1. **delete (startindex [, endindex])** : cette méthode supprime un caractère spécifique ou une plage de texte.
2. **get (startindex [, endindex])** : cette méthode retourne un caractère spécifique ou une plage de texte.
3. **index (index)** : retourne la valeur absolue d'un index basé sur l'index donné.
4. **insert (index [, string] ...)** : cette méthode insère des chaînes à l'emplacement d'index spécifié.
5. **see(index)** : cette méthode retourne `true` si le texte situé à la position d'index est visible.
6. **mark_gravity (mark [, gravity])** : retourne la gravité de la marque donnée. Si le deuxième argument est fourni, la gravité est définie pour la marque donnée.
7. **mark_names ()** : retourne toutes les marques du widget Texte.
8. **mark_set (mark, index)** : informe une nouvelle position par rapport à la marque donnée.
9. **mark_unset (mark)** : supprime la marque donnée du widget Texte.

10. **tag_add (tagname, startindex [, endindex] ...)** : cette méthode balise la position définie par startindex ou une plage délimitée par les positions startindex et endindex.
11. **tag_config()** : Vous pouvez utiliser cette méthode pour configurer les propriétés de la balise, qui comprennent, justifier (centre, gauche ou droite), des onglets (cette propriété a les mêmes fonctionnalités que la propriété des onglets du widget Texte) et un soulignement (utilisé pour souligner le texte marqué).).
12. **tag_delete (tagname)** : cette méthode est utilisée pour supprimer une balise donnée.
13. **tag_remove (tagname [, startindex [.endindex]] ...)** : après avoir appliqué cette méthode, la balise donnée est supprimée de la zone fournie sans supprimer la définition de balise réelle.

Exemple. Tkinter Text Widget

```

1 # coding : utf-8
2 from tkinter import *
3 root= Tk()
4 t = Text(root , fg="red" , padx=50)
5 t.config(font=("broadway" , 14))
6 t.insert(1.0 , "Exemple de widget Text Tkinter ")
7 t.pack()
8 root.mainloop()

```

Ce qui affiche après exécution :



7.2.6 Le widget Frame Tkinter

Le widget **Frame (cadre)** est très important pour le processus de regroupement et d'organisation des autres widgets de manière conviviale.

Cela fonctionne comme un conteneur, qui est responsable de la position des autres widgets.

Il utilise des zones rectangulaires à l'écran pour organiser la mise en page et fournir un remplissage de ces widgets. Un **Frame** peut également être utilisé comme classe de base pour implémenter des widgets complexes.

7.2.6.1 Syntaxe

```
1 w = Frame (master, option, ...)
```

1. **master** : représente la fenêtre ou l'objet parent.
2. **options** : liste des options les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

7.2.6.2 Liste des options d'un widget Frame

1. **bg** : couleur d'arrière-plan du widget Frame
2. **bd** : taille des bordures autour du Frame.
3. **cursor** : permet de personnaliser le motif du curseur de la souris au moment du survole.
4. **height** : définit la dimension verticale du Frame.
5. **highlightbackground** : définit la couleur de la mise au point en surbrillance de l'objet Frame.
6. **highlightcolor** : couleur de surbrillance lorsque le frame a le focus.
7. **relief** : avec la valeur par défaut **relief = FLAT**, la case à cocher ne ressort pas de son arrière-plan. Vous pouvez définir cette option sur n'importe quel autre style.
8. **width** : définit la largeur du frame

Exemple. Frame Tkinter

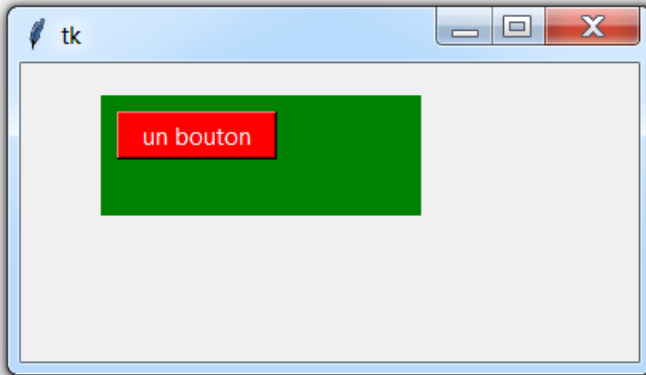
```
1 # coding : utf-8
2 from tkinter import *
3
4 # Création de la fenêtre principale
5 master = Tk()
6 master.geometry("400x200")
7
8 #Création d'un frame d'arrière plan vert
9 frm = Frame(master, bg='green')
10
```

```

11 # Emplacement du frame
12 frm.place(x=50, y=20, width=200,height=75)
13
14 # Création d'un bouton au sein du frame
15 b = Button(frm, text="un bouton",bg='red', fg='white')
16 b.place(x=10, y=10, width=100,height=30)
17 master.mainloop()

```

Ce qui affiche après exécution :



7.2.7 Le widget Scale Tkinter

Le widget **Scale** permet à l'utilisateur de **sélectionner une valeur numérique** en **déplaçant un bouton curseur** le long d'une échelle. Vous pouvez contrôler les valeurs minimales et maximales, ainsi que la résolution.

Quand utiliser le widget Scale

Vous pouvez utiliser le widget **Scale** au lieu d'un widget Entry, lorsque vous souhaitez que l'utilisateur saisisse une valeur numérique limitée.

Syntaxe :

```
1 nom_du_Scale = Scale(Fenetre_principale , options)
```

Exemple. .

```

1 from tkinter import *
2
3 def select():
4     sel = "Value = " + str(v.get())

```

```

5     print(sel)
6
7     root = Tk()
8     root.geometry("400x300")
9     v = IntVar()
10    scale = Scale( root , variable = v, from_ = 1, to = 30 ,
11                  orient = HORIZONTAL )
12    #scale.bind("<B1-Motion>",select)
13    scale.pack(anchor=CENTER)
14
15    button = Button(root , text="Get Scale Value" , command=
16                    select)
17    button.pack(anchor=CENTER)
18
19    root.mainloop()

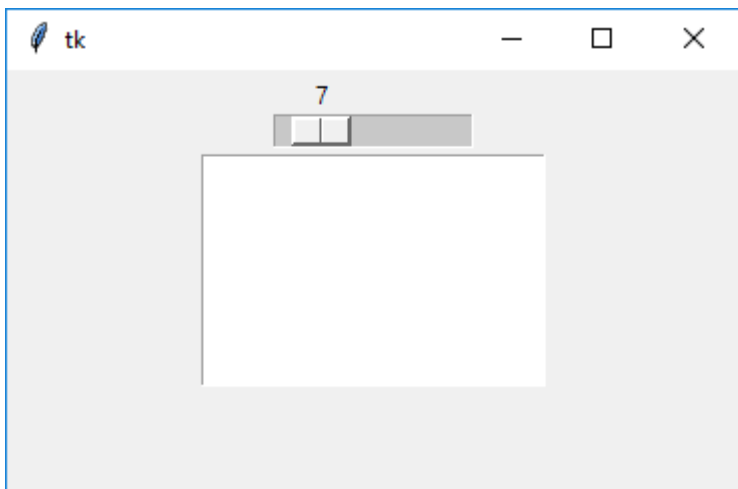
```

Maintenant, pour bien apprécier l'utilité du **widget Scale**, nous allons traiter un exemple de ce dernier lié à un **événement 7.7**

```

1  # coding : utf-8
2  from tkinter import *
3
4  # méthode qui réalise l'action associée au widget scale
5  def select(event) :
6      # récupération de la valeur du scale
7      sel = "Value = " + str(v.get())
8      # redimensionnement du widget Text
9      T.configure(width=3*v.get() , height = v.get())
10     T.pack()
11
12     root = Tk()
13     root.geometry("700x500")
14     v = IntVar()
15     scale = Scale( root , variable = v, from_ = 1, to = 50,
16                   orient = HORIZONTAL)
17     # Lier un événement bind au mouvement du widget scale
18     scale.bind("<B1-Motion>",select)
19     scale.pack(anchor=CENTER)
20
21     #Création d'un widget Text qui sera manipulé par le
22     # widget Scale
23     T = Text(root )
24     T.configure(width=10 , height = 5)
25     T.pack()
26
27     root.mainloop()

```

Vous pouvez constater maintenant le **redimensionnement** de la **zone** du **texte** quand on fait varier la **position** du **curseur**.

7.3 Les attributs standard des widgets Tkinter

Malgré la diversité de leurs méthodes et propriétés, les **widgets Tkinter** possèdent des attributs communs, tels que la taille, la couleur et la police sont spécifiés :

1. **Dimensions**
2. **Colors**
3. **Fonts**
4. **Anchors**
5. **Relief styles**
6. **Bitmaps**
7. **Cursors**

Exemple. Label avec couleur de font et background personnalisés

```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3 #Création d'une fenêtre Tkinter
4 f = Tk()
```

```

5 f.geometry("300x75")
6 #Création d'un widget label de couleur blanche, bold,
   taille 18...
7 Centre = Label(f, text = "CRMEF OUJDA ", bg="black", fg="
   white", font='broadway 18 bold')
8 Centre.pack()
9 f.mainloop()

```

Ce qui affiche après exécution :



7.4 Les méthodes de gestion des dispositions géométriques des widgets

Tous les **widgets Tkinter** ont accès aux méthodes de gestion de géométrie spécifiques, qui ont pour but d'organiser les widgets dans la zone du widget parent. Tkinter possède les classes de gestionnaire de géométrie suivantes :

1. La méthode `pack()`
2. La méthode `grid()`
3. La méthode `place()`

7.4.1 La méthode de disposition géométrique `pack()`

Le gestionnaire de disposition géométrique **`pack()`**, organise les widgets en blocs avant de les placer dans le widget parent. Les widgets sont placés l'un au dessous de l'autre selon l'ordre d'application de la méthode `pack()` avec un emplacement **centré par défaut**.

Exemple. usage de la méthode `pack()`

```

1 # -*- coding : utf-8 -*-
2 from tkinter import *
3
4 #Création d'une fenêtre Tkinter

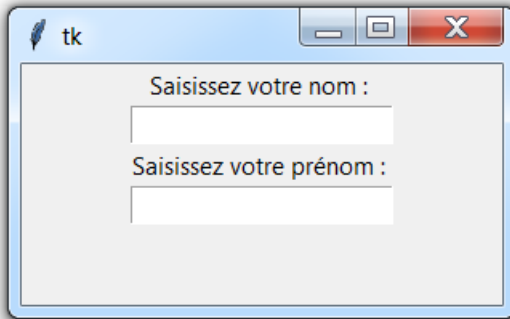
```

```

5 f = Tk()
6 f.geometry("300x150")
7 Nom = Label(f, text="Saisissez votre nom : ")
8 Prenom = Label(f, text="Saisissez votre prénom : ")
9 champNom = Entry(f) champPrenom = Entry(f)
10 Nom.pack()
11 champNom.pack()
12 Prenom.pack()
13 champPrenom.pack()
14 f.mainloop()

```

Ce qui affiche après exécution :



7.4.2 La méthode de disposition géométrique `grid()`

Le gestionnaire de disposition géométrique `grid()`, organise les widgets dans une structure de type table dans le widget parent.

Exemple. usage de la méthode `grid()`

```

1 # -*- coding : utf-8 -*-
2 from tkinter import *
3
4 #Création d'une fenêtre
5 Tkinter f = Tk()
6 f.geometry("350x100")
7
8 #Création des widgets
9 Nom = Label(f, text="Saisissez votre nom : ")
10 Prenom = Label(f, text="Saisissez votre prénom : ")
11 champNom = Entry(f)
12 champPrenom = Entry(f)
13
14 #Application de la méthode grid() aux widget
15 Nom.grid(row=0, column=0)

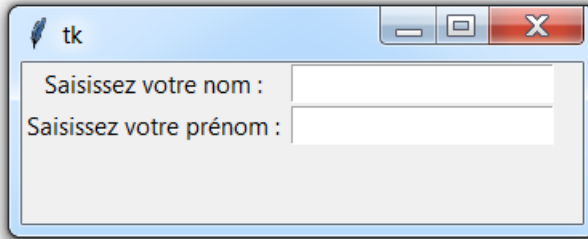
```

```

16 Prenom.grid(row=1, column=0)
17 champNom.grid(row=0, column=1)
18 champPrenom.grid(row=1, column=1)
19 f.mainloop()

```

Ce qui affiche après exécution :



7.4.3 La méthode de disposition géométrique `place()`

Le gestionnaire de disposition géométrique **`place()`**, organise les widgets en les plaçant à des positions spécifiques dans le widget parent suivant leurs coordonnées et leurs dimensions :

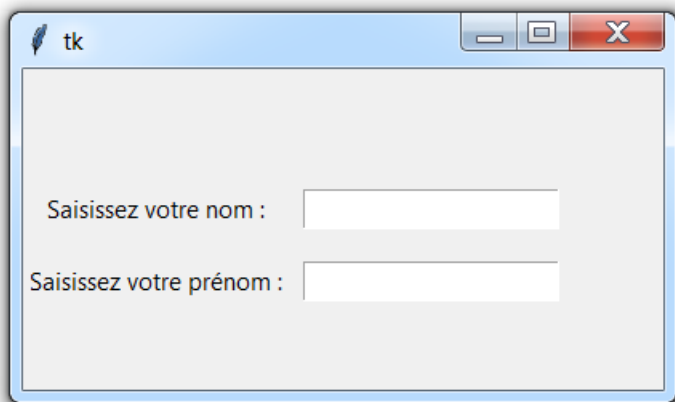
Exemple. Usage de la méthode **`place()`** :

```

1  # -*- coding : utf-8 -*-
2  from tkinter import *
3
4  #Création d'une fenêtre Tkinter
5  f = Tk()
6  f.geometry("400x200")
7
8  #Création des widgets
9  Nom = Label(f, text = "Saisissez votre nom : ")
10 Prenom = Label(f, text="Saisissez votre prénom : ")
11 champNom = Entry(f)
12 champPrenom = Entry(f)
13 #Application de la méthode place() aux widget
14 Nom.place(x=5, y=75, width=160, height=25)
15 champNom.place(x=175, y=75, width=160, height=25)
16 Prenom.place(x=5, y=120, width=160, height=25)
17 champPrenom.place(x=175, y=120, width=160, height=25)
18 f.mainloop()

```

Ce qui affiche après exécution :



7.5 Actions manipulant des widgets Tkinter

7.5.1 Action associée à un bouton de commande

7.6 Menu Tkinter en Python

7.6.1 La classe Menu Tkinter

Le rôle du **widget Menu** est de vous permettre de créer toutes sortes de menus utilisables par vos applications. La création d'un **menu Tkinter** est réalisable à l'aide de la **classe Menu** et se déroule selon les étapes suivantes :

7.6.2 Création de la barre des menus

```
1 menuBar = Menu (master) # master designe la fenêtre principale
```

7.6.3 Création d'un menu principal

Pour créer un **menu principal**, il suffit d'instancier la **classe Menu** en prenant comme **paramètre** l'**objet menuBar** et d'appliquer la méthode **add_cascade()** pour ajouter des label. Voici comment créer un menu principal 'Fichier'

```

1 menuFichier = Menu(menuBar, tearoff = 0)
2 #tearoff = 0 pour menu non détachable, tearoff=1 pour menu
  détachable
3 menuBar.add_cascade(label = "Fichier", menu=menuFichier)

```

7.6.4 Création des commandes ou sous menu du menu principal

Pour ajouter une commande au menu principal, on applique la méthode `add_command()`. Exemple si on veut ajouter la **commande Nouveau** au menu **Fichier** :

```

1 menuFichier.add_command(label="Nouveau")

```

Et si de plus on veut associer une action à la commande :

```

1 menuFichier.add_command(label="Nouveau" , command = [
  nom_de_la_commande])

```

7.6.5 Configuration de la barre des menus

Finalement pour configurer le menu, avec la fenêtre principale, on applique la méthode `config()`

```

1 master.config(menu = menuBar)

```

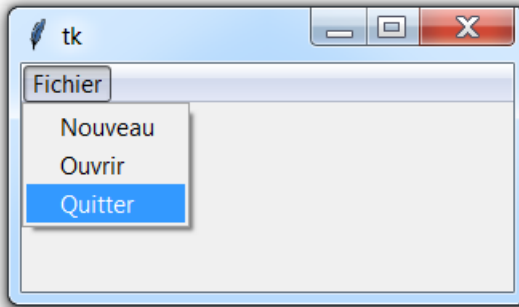
Exemple. menu Fichier

```

1 # coding : utf-8
2 from tkinter import *
3 master = Tk()
4
5 # Création de la barre des menu
6 menuBar = Menu(master)
7
8 # Création du menu principal 'Fichier'
9 menuFichier = Menu(menuBar, tearoff = 0)
10 menuBar.add_cascade(label="Fichier", menu=menuFichier)
11
12 # Création des sous menus : 'Nouveau', 'Ouvrir', 'Quitter'
13 menuFichier.add_command(label="Nouveau")
14 menuFichier.add_command(label="Ouvrir")
15 menuFichier.add_command(label="Quitter", command=quit)
16
17 # Configuration de la barre des menus
18 master.config(menu=menuBar)
19 master.mainloop()

```

Enregistrons le fichier sous un nom à titre d'exemple : `menu_fichier.py` et exécutons le :



7.6.6 Action associée à une commande d'un menu

Nous allons traiter cela sur un exemple simple : la création de l'action associée à la **commande** *Fichier* → *Nouveau*. Nous devons donc ajouter l'option **command** :

```
1 menuFichier.add_command(label="Nouveau" , command = new)
```

Nous devons ensuite créer la méthode **new** qui va réaliser l'action :

```
1 def new() :
2     ...
```

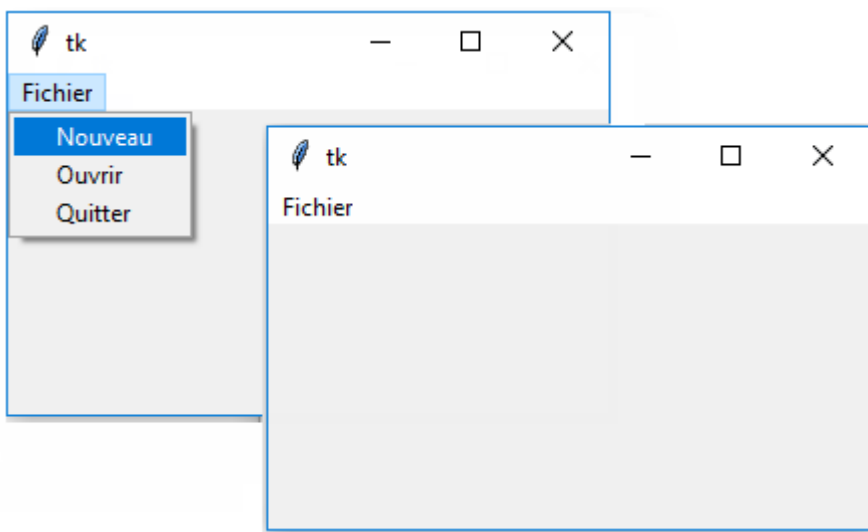
Créons un **fichier python nommé** à titre d'exemple : `menu_fichier.py` et exécutons le :

```
1 # coding : utf-8
2 from tkinter import *
3 import os
4
5 def new() :
6     # on exécute le même fichier menu_fichier.py
7     os.popen("python menu_fichier.py")
8
9 master = Tk()
10 master.geometry("300x150")
11 # Création de la barre des menu
12 menuBar = Menu(master)
13
14 # Création du menu principal 'Fichier'
15 menuFichier = Menu(menuBar, tearoff = 0)
16 menuBar.add_cascade(label="Fichier", menu = menuFichier)
```

```

17
18 # Création des sous menus : 'Nouveau', 'Ouvrir', 'Quitter'
19 menuFichier.add_command(label = "Nouveau" , command = new)
20 menuFichier.add_command(label = "Ouvrir")
21 menuFichier.add_command(label = "Quitter" , command = quit)
22
23 # Configuration de la barre des menus
24 master.config(menu = menuBar)
25 master.mainloop()

```



Remarquez bien quand on clique sur la commande **Fichier** → **Nouveau**, une nouvelle fenêtre est créée !

7.7 Les événements en Tkinter (binding event)

7.7.1 Lier un événement à un widget

Les événements en Tkinter sont nombreux et très variés et peuvent provenir de diverses sources, y compris les pressions sur les touches du clavier, les opérations de la souris : bouton gauche, bouton droit, roulette...

Tkinter fournit un mécanisme puissant vous permettant de produire et de gérer vous-même vos propres événements. Pour chaque widget, vous pouvez lier des méthodes Python produisant un ou plusieurs événements.

Syntaxe :

```
1 widget.bind(event, handler)
```

1. **event** : est l'**action utilisateur** comme un click de souris, appui sur une touche du clavier, mouvement de la souris...
2. **handler** : est le **gestionnaire** de l'**événement** : c'est à dire la méthode qui sera appelée en réponse à l'action utilisateur.

Voici un exemple simple :

Exemple. Position de la souris au moment du clics dans une fenêtre

```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3 master = Tk()
4
5 # méthode qui gère l'événement
6 def callback(event) :
7     print("Vous avez cliquez à la position : ", event.x,
8         event.y)
9
10 # Associer l'événement au click gauche de la souris
11 master.bind("<Button-1>", callback)
12
13 master.mainloop()
```

Exemple. récupération de la touche appuyée sur le clavier

```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3
4 master = Tk()
5
6 # méthode de récupération de la position du click
7 def action(event) :
8     frm.focus_set()
9     print("Vous avez cliqué à la position : ", event.x,
10         event.y)
11
12 #méthode de récupération de la touche appuyée sur le
13     clavier
14 def key(event) :
15     print("Vous avez appuyé sur la touche : ", repr(event.
16         char))
```

```

15 frm = Frame(master, width=200, height=200)
16 frm.bind("<Key>", key)
17 frm.bind("<Button-1>", action)
18 frm.pack()
19
20 master.mainloop()

```

Après exécution du programme si on clique à une position de la fenêtre et on appui sur la **touche 'a'** du clavier on verra l'affichage du message :

Vous avez cliqué à la position : 27 26

Vous avez appuyé sur la touche : 'a'

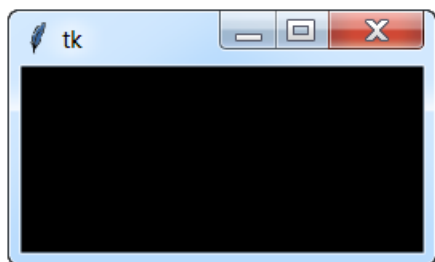
7.7.2 Les différents types d'événements Tkinter

1. **<Button-1>** : événement associé au **bouton gauche** de la souris.
2. **<ButtonPress>** : événement associé à la fois au bouton gauche et droit de la souris.
3. **<B1-Motion>** : événement associé au **déplacement** de la souris avec bouton gauche **enfoncé**.
4. **<ButtonRelease-1>** : événement associé au **relâchement** du bouton gauche de la souris.
5. **<Double-Button-1>** : événement associé au **double-click** du bouton gauche de la souris.
6. **<Enter>** : événement qui se produit lorsque le pointeur de la souris est entré dans le widget (cet événement ne signifie pas que l'utilisateur a appuyé sur la touche Entrée!).
7. **<Leave>** : événement qui se produit lorsque le pointeur de la souris quitte le widget
8. **<FocusIn>** : événement qui se produit lorsque le focus du clavier a été déplacé vers ce widget.
9. **<FocusOut>** : événement qui se produit lorsque le focus du clavier quitte le widget.
10. **<Return>** : événement qui se produit lorsque l'utilisateur a appuyé sur la touche Entrée. Vous pouvez lier à pratiquement toutes les touches du clavier : **Cancel**, **BackSpace**, **Échap**, **F1**, **F2**, **F3**...
11. **<Key>** : événement qui se produit lorsque l'utilisateur a appuyé sur n'importe quelle touche du clavier.
12. **<Any-KeyPress>** :

Exemple. changement de couleur du background d'une fenêtre Tkinter

```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3
4 def action1(event) :
5     root.configure(bg = "white")
6
7 def action2(event) :
8     root.configure(bg = "black")
9
10 root = Tk()
11 root.geometry("200x100")
12
13 root.bind("<Enter>" , action1)
14 root.bind("<Leave>" , action2)
15 root.mainloop()
```

L'exemple ci-dessus permet de changer la couleur du background en noir lorsque la souris quitte la fenêtre et en couleur blanche lorsque la souris entre dans la fenêtre.

**Événement FocusIn****Événement FocusOut**

7.7.3 L'objet event

L'objet **event** est une instance d'objet Python standard, ayant un certain nombre d'attributs décrivant l'événement qui se produit.

7.7.3.1 Attributs de l'objet event

1. **L'attribut `widget`** : indique le widget qui a **produit l'événement** : comme bouton, label, zone de saisie...
2. **L'attribut `x, y`** : détermine la position actuelle de la souris en pixels sur la fenêtre. Le coin gauche supérieur de la fenêtre correspond à $(x, y) = (0, 0)$.
3. **L'attribut `x_root, y_root`** : détermine la position actuelle de la souris par rapport au coin supérieur gauche de l'écran, en pixels.

4. **L'attribut char** : indique le caractère de la touche clavier utilisée (événements clavier uniquement), sous forme de chaîne.
5. **L'attribut keysym** : indique le key symbol de la touche système utilisée (événements clavier uniquement).
6. **L'attribut keycode** : code de touche utilisée (événements clavier uniquement).
7. **L'attribut num** : numéro du bouton (événements de bouton de souris uniquement).
8. **L'attribut width, height** : indique la nouvelle largeur et hauteur du widget (seulement pour un événement Configure)
9. **L'attribut type** : indique le type d'événement. Exemple pour une touche de clavier cet attribut renvoie : **EventType.KeyPress**

Exemple. ...

```

1  # -*- coding : utf-8 -*-
2  from tkinter import *
3
4  def action(event) :
5      print((event.char ,
6              event.keysym ,
7              event.keycode ,
8              event.type))
9      print("La position de la souris est : " ,(event.x ,
10         event.y))
11
12 root = Tk()
13 root.geometry("300x200")
14 root.bind("<Any-KeyPress>" , action)
15
16 root.mainloop()
```

Après exécution du code en pointant sur un endroit de la fenêtre avec le pointeur de la souris sans faire aucun click et en appuyant sur la touche '=' du clavier on obtiendra le message à l'écran :

('=', 'equal', 187, < EventType.KeyPress : '2' >)

La position de la souris est : (74, 69)

7.8 La bibliothèque d'images Pillow

7.8.1 La bibliothèque Pillow

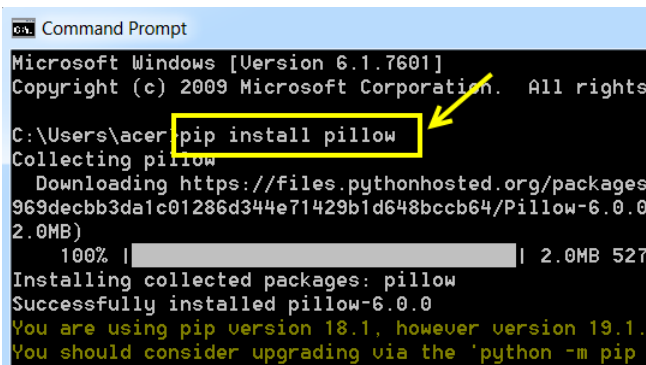
Pour traiter des images, Python dispose d'une librairie nommée **Pillow**. Pillow constitue actuellement un fork successeur du projet **PIL (Python**

Imaging Library). Elle est conçue de manière à offrir un accès rapide aux données contenues dans une image, elle est dotée d'un pouvoir magique et puissant pour le traitement et la manipulation des différents formats de fichiers images tels que **PNG, JPEG, GIF, TIFF et BMP**.

7.8.2 Installation de la bibliothèque Pillow

La bibliothèque Pillow s'installe d'une façon très simple à l'aide de l'utilitaire **pip** :

```
1 pip install pillow
```



```

C:\Users\acer> Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\acer> pip install pillow
Collecting pillow
  Downloading https://files.pythonhosted.org/packages/96/9d/ec/bb/3da1c01286d344e71429b1d648bccb64/Pillow-6.0.0-2.0MB)
    100% |#####| 2.0MB 527KB/s
Installing collected packages: pillow
Successfully installed pillow-6.0.0
You are using pip version 18.1, however version 19.1.0 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
  
```

7.8.3 Insertion d'image à l'aide de la bibliothèque Pillow

Maintenant que la bibliothèque Pillow est installé, elle peut être utilisée pour insérer et manipuler des images au sein d'une fenêtre Tkinter. Pour ce faire, il suffit de l'importer et de créer un objet image à partir d'un fichier :

```

1 from PIL import Image, ImageTk
2
3 # Création de l'objet image
4 load = Image.open("chemin_vers_le_fichier_images")
5
6 # Création de la photo à partir de l'objet image
7 photo = ImageTk.PhotoImage(load)
  
```

Pour bien comprendre, nous allons traiter cela sur un exemple simple :

Considérons une image nommée **voiture.png** qui se trouve dans un dossier **images**

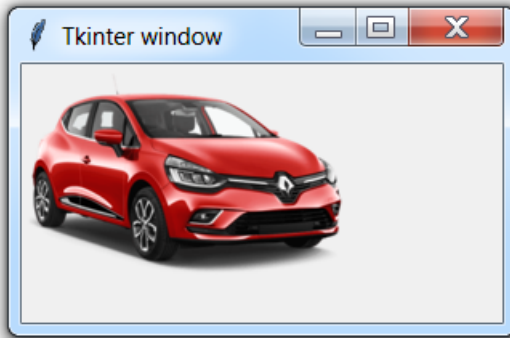
Exemple. Insertion d'image

```

1 from tkinter import *
2 from PIL import Image, ImageTk
3
4 root = Tk()
5 root.title("Tkinter window")
6 root.geometry("300x200")
7
8 # Création de l'objet image
9 load = Image.open("images/voiture.png")
10
11 # Création de la photo à partir de l'objet image
12 photo = ImageTk.PhotoImage(load)
13
14 # Placer l'image dans un label
15 label_image = Label(root, image=photo)
16 label_image.place(x=0, y=0)
17 root.mainloop()

```

Ce qui affiche après exécution :

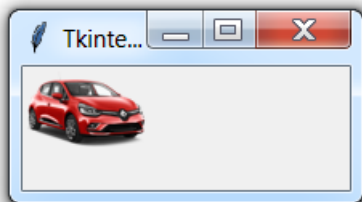


Remarque 17. On peut redimensionner l'image à l'aide la méthode **thumbnail()**

```

1 # Création de l'objet image
2 load = Image.open("images/voiture.png")
3
4 # Redimensionner l'image
5 load.thumbnail((50,50))

```



7.8.4 Convertir une image avec Pillow

La bibliothèque **Pillow**, permet de convertir une image d'un format déterminé à un autre format souhaité d'une façon assez rapide via la méthode **convert()** :

Exemple. convertir une image au format png au format jpg

```

1 # importation de la bibliotheque Pillow
2 from PIL import Image
3
4 # Ouvrir une image au format png
5 im = Image.open("monImage.png")
6
7 # Charger la conversion
8 rgb_im = im.convert('RGB')
9
10 # obtenir une image au format jpg
11 rgb_im.save('monImage.jpg')
```

7.9 Les boîtes de dialogues en Tkinter

La bibliothèque Tkinter est dotée de trois principaux sous-modules destinés spécialement à la création des boîtes de dialogues dont le but d'interagir avec l'utilisateur :

1. **tkinter.messagebox** : affiche des messages d'informations, alert, de confirmation...
2. **tkinter.filedialog** : affiche une boîte d'exploration des fichiers et dossiers.
3. **tkinter.colorchooser** : affiche une boîte donnant à l'utilisateur la possibilité de sélectionner une couleur. Afin de pouvoir utiliser un de ces module, il faut préalablement l'importer :

Exemple. importation du module messagebox


```

1 from tkinter import *
2 from tkinter import messagebox

```

7.9.1 Le module messagebox

Le module **messagebox** est utilisé pour afficher des boîtes de message dans vos applications. Ce module fournit un certain nombre de fonctions que vous pouvez utiliser pour afficher un message approprié.

Les méthodes associées à ce module, sont :

1. **showinfo()**,
2. **showwarning()**,
3. **showerror()**,
4. **askquestion()**,
5. **askokcancel()**,
6. **askyesno()**,
7. **askretrycancel()**.

Syntaxe :

```

1 messagebox.FunctionName(title , message [ , options ])

```

Paramètres

- **FunctionName** : Il s'agit du nom de la fonction de boîte de message.
- **title** : Il s'agit du texte à afficher dans la barre de titre de la boîte de message.
- **message** : Il s'agit du texte à afficher sous forme de message.
- **options** : les options sont des choix alternatifs que vous pouvez utiliser pour personnaliser une boîte de message standard. Certaines des options que vous pouvez utiliser sont par défaut et parent. L'option par défaut est utilisée pour spécifier le bouton par défaut, comme **ABORT**, **RETRY** ou **IGNORE** dans la boîte de message. L'option parent est utilisée pour spécifier la fenêtre au-dessus de laquelle la boîte de message doit être affichée.

Nous allons voir maintenant quelques exemples d'usage de la boîte de dialogue **messagebox** :

Exemple. Quand l'utilisateur clique par erreur! sur le bouton **Quitter**, il faut à ce moment là coder correctement l'application afin de l'avertir s'il veut vraiment quitter l'application. Faute de quoi, il peut perdre des données précieuses!

```

1  # -*- coding : utf-8 -*-
2  from tkinter import *
3  from tkinter.messagebox import*
4
5  def action() :
6      reponse = askyesnocancel("Terminer le script",
7                              "Voulez vous vraiment quitter
8                              ? \n cliquer   ou  z pour finir")
9      if reponse :
10         root.quit()
11
12 root = Tk()
13 root.geometry("300x200")
14
15 bQuitter = Button(root , text = "Quitter" , command =
16                 action)
17 bQuitter.pack()
18
19 root.mainloop()

```

Exemple. En cr  ant des quiz, vous souhaitez afficher dans le cas o   la r  ponse est correcte, le message '**bonne r  ponse!**' et en cas o   la r  ponse est fausse, un **message d'erreur!** suivie d'une boite '**Rerty**' lui proposant de **r  essayer** encore une fois.

```

1  # -*- coding : utf-8 -*-
2  from tkinter import *
3  from tkinter import messagebox
4  from tkinter.messagebox import*
5
6  def action() :
7      if (E.get() == "Madrid") :
8          messagebox.showinfo("Information","Bonne r  ponse !"
9                              ")
10         else :
11             messagebox.showerror("Error", "Mauvaise r  ponse!")
12             msg = messagebox.askretrycancel(
13                 "Question",
14                 "La r  ponse est fausse ! \n Voulez-vous
15                 r  essayer ?")
16             if(not msg) :
17                 root.quit()
18
19 root = Tk()
20 root.geometry("300x200")
21 L = Label(root , text = "Quelle est la capitale de l'
22         espagne? ")
23 L.pack()
24 E = Entry(root)
25 E.pack()

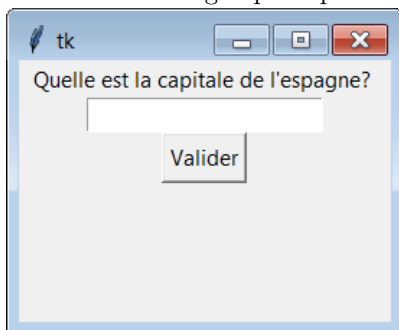
```

```

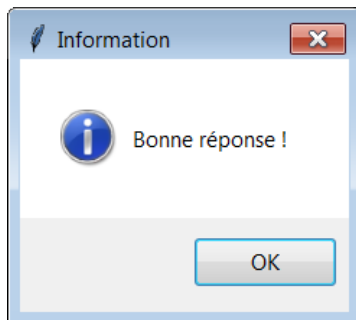
23 bValider = Button(root , text = "Valider" , command =
    action)
24 bValider.pack()
25
26 root.mainloop()

```

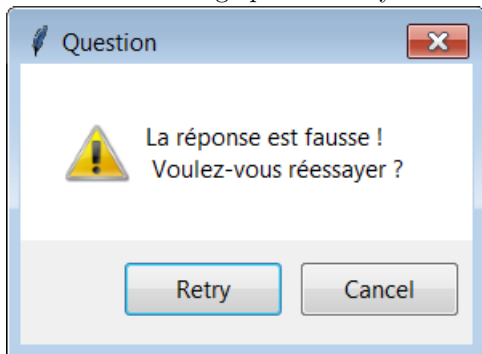
Boite de dialogue principale



Boite message : réponse juste



Boite message pour réessayer



7.9.2 Le module colorchooser

7.9.3 Le module Filedialogue

7.9.3.1 Le module Filedialogue et les méthodes associées

FileDialog est un module avec des fonctions de dialogue d'ouverture et de sauvegarde, qui pourra vous aider à améliorer votre interface **graphique Tkinter** pour ouvrir ou sauvegarder vos fichiers.

Le module **FileDialog** est doté de trois principale méthodes :

1. **askopenfilename()** : affiche une **boite de dialogue** qui demande l'**ouverture d'un fichier** existant.

2. **asksaveasfilename()** : affiche une **boîte de dialogue** qui permet d'enregistrer un fichier
3. **askdirectory()** : affiche une **boîte de dialogue** qui demande l'ouverture d'un répertoire.

7.9.3.2 La méthode askopenfilename

Comme nous l'avons expliqué ci haut, cette méthode affiche une boîte de dialogue qui demande de sélectionner le fichier à ouvrir et récupère le fichier avec son chemin sur une variable. Elle possède les paramètres suivants :

1. **initialdir** : qui permet de pointer vers un répertoire initial dès son ouverture.
2. **title** : qui permet de personnaliser le titre de la boîte de dialogue
3. **filetypes** : qui permet de spécifier les types de fichiers à ouvrir

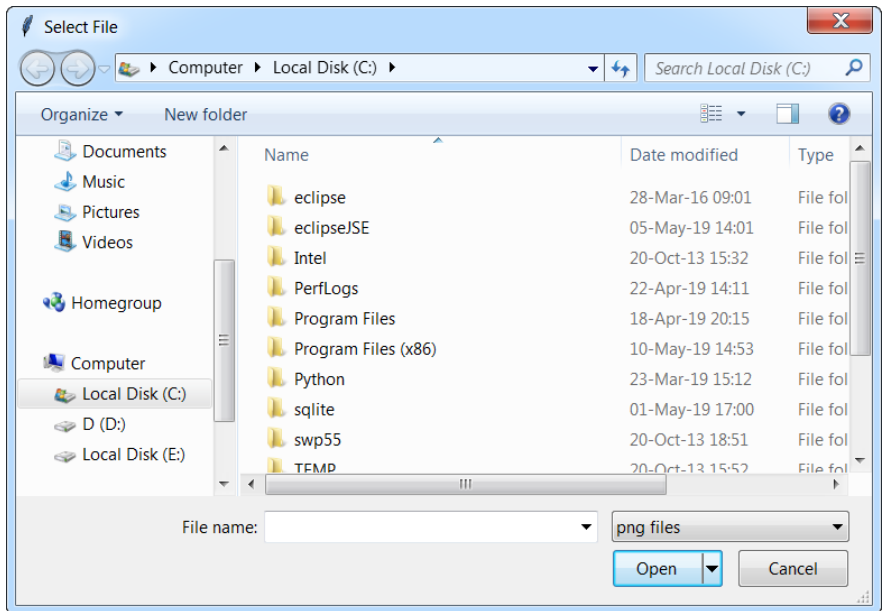
Exemple. methode askopenfilename

```

1 from tkinter import filedialog
2 from tkinter import *
3 root = Tk()
4 filename = filedialog.askopenfilename(initialdir= "/",
    title="Select File",filetypes=(("png files", "*.png"),(
    "jpeg files", "*.jpg"),("all files", "*.*")))
5 print(filename)

```

Ce qui permet de lancer la boîte de dialogue suivante :



7.9.3.3 La méthode `asksaveasfilename`

Cette méthode est semblable à la précédente à la différence qu'elle enregistre un fichier au lieu de l'ouvrir et elle possède donc exactement les mêmes paramètres

Exemple. méthode `asksaveasfilename`

```

1 from tkinter import filedialog
2 from tkinter import *
3 root = Tk()
4 filename = filedialog.asksaveasfilename(initialdir = "/",
5     title = "Select file", filetypes = (("png files", "*.png"),
6     ("jpeg files", "*.jpg"), ("all files", "*.*")))
7 print(filename)

```

7.9.3.4 La méthode `askdirectory`

Comme nous l'avons déjà mentionné, cette méthode demande à l'utilisateur de choisir le répertoire à ouvrir. Quand l'utilisateur sélectionne

un répertoire ce dernier sera stocké dans une variable du type `string` qui contient le chemin vers le répertoire sélectionné.

Exemple. méthode `askdirectory`

```
1 from tkinter import *
2 from tkinter import filedialog
3 root = Tk()
4 myDirectory = filedialog.askdirectory()
5 print (myDirectory)
```

7.10 Le module de design `tkinter.ttk`

7.10.1 A propos du module `tkinter.ttk`

Le module **`tkinter.ttk`** est une **extension** de la bibliothèque **`Tkinter`** qui fournit un accès au jeu de **style** pour les widgets **`Tk`**.

Les widgets dans `tkinter` sont hautement et facilement configurables. Vous avez un contrôle presque total sur leur apparence : largeurs de bordure, polices, images, couleurs, etc. Mais avec un style pauvre et très basique.

Les **widgets `ttk`** utilisent des styles pour définir une apparence et un look agréable. Il faut donc un peu plus de travail si vous souhaitez un bouton non standard. Les widgets `ttk` sont également un peu sous-documentés.

En règle générale, les **widgets à thème `ttk`** vous donneront une application plus "native", mais aux dépens d'une perte de configurabilité.

Il est conseillé d'utiliser les **widgets `ttk`** si vous voulez que votre interface graphique apparaisse agréable et un peu plus **moderne**, et les **widgets `tkinter`** si vous avez besoin d'un peu plus de configurabilité. Vous pouvez les utiliser tous les deux dans la même application.

7.10.2 Usage du module `tkinter.ttk`

Afin de pouvoir utiliser le **module `ttkinter.ttk`**, il faut préalablement **l'importer** en même temps que la **bibliothèque `Tkinter`** :

```
1 from tkinter import ttk
2 from tkinter import *
3 from tkinter.ttk import *
```

7.10.3 Usage de tkinter.ttk Button, Label & Entry

Afin de pouvoir utiliser le style du **module ttk**, il faut au préalable créer un style en appelant la **méthode style()** de la **classe ttk** :

```
1 style = ttk.Style()
2 style.configure(...)
```

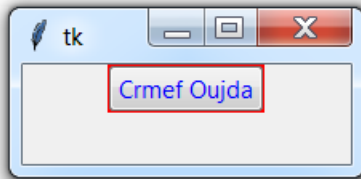
Exemple. Création d'un ttk style pour un bouton

```
1 style = ttk.Style()
2 style.configure("BW.TButton", foreground="blue",
                 background="red")
```

Code complet :

```
1 from tkinter import ttk
2 from tkinter import *
3 from tkinter.ttk import *
4 root = Tk()
5 style = ttk.Style()
6 style.configure("BW.TButton", foreground="blue",
                 background="red")
7 b = ttk.Button(text = "Crmeff Oujda", style="BW.TButton")
8 b.pack()
9 root.mainloop()
```

Ce qui affiche à l'exécution :



Exemple. style ttk pour un label

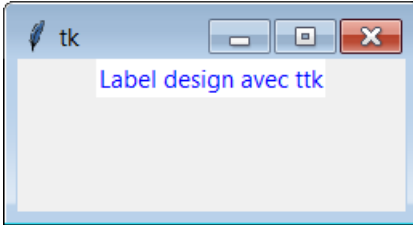
```
1 # coding : utf-8
2 from tkinter import ttk
3 from tkinter import *
4 from tkinter.ttk import *
5 root = Tk()
6 root.geometry("300x200")
7
8 # définir le style
9 style = ttk.Style()
```

```

10 style.configure("BW.TLabel", foreground="blue", background
    ="white")
11 # Label avec un style BW.TLabel
12 L = ttk.Label(root , text ="Label design avec ttk", style
    = "BW.TLabel")
13 L.pack()
14
15 root.mainloop()

```

Ce qui donne :



7.10.4 Usage de ttk.Combobox

L'objet Combobox du module ttk Tkinter, permet de créer des listes déroulantes avec plusieurs options.

Syntaxe :

```

1 combo = ttk.Combobox(fenêtre_principale , values =
    liste_valeurs )

```

Exemple. Liste Combobox

```

1 # -*- coding : utf-8 -*-
2 import tkinter as tk
3 from tkinter import ttk
4
5 # Fenêtre principale
6 root = tk.Tk()
7 root.geometry('300x200')
8 # Label au top
9 labelTop = tk.Label(root , text = "Choisissez votre Laptop"
    )
10 labelTop.pack()
11
12 # Liste des valeurs d'option de la combobox
13 liste_valeurs=["Acer", "HP", "Del", "Asus"]
14
15 # Création la liste combobox
16 combo = ttk.Combobox(root , values = liste_valeurs )
17 combo.pack()

```



```

18 # Définir l'élément qui s'affiche par défaut
19 combo.current(0)
20
21 root.mainloop()

```

Remarque 18. On peut aussi associer une action liée (**bind action**) en utilisant la commande :

```

1 combo.bind("<<ComboboxSelected>>", comboAction)

```

Exemple. Code complet de la liste Combo

```

1 # -*- coding : utf-8 -*-
2 import tkinter as tk
3 from tkinter import ttk
4
5 # Définir l'action liée (bind action)
6 def comboAction(event) :
7     select = combo.get()
8     print("Vous avez choisie : ", select)
9 # Fenêtre principale
10 root = tk.Tk()
11 root.geometry('300x200')
12 # Label au top
13 labelTop = tk.Label(root, text = "Choisissez votre Laptop"
14 )
15 labelTop.pack()
16
17 # Liste des valeurs d'option de la combobox
18 liste_valeurs=["Acer", "HP", "Del", "Asus"]
19
20 # Création la liste combobox
21 combo = ttk.Combobox(root, values = liste_valeurs )
22 combo.pack()
23 # Définir l'élément qui s'affiche par défaut
24 combo.current(0)
25 # Associé une bind action à la liste combo
26 combo.bind("<<ComboboxSelected>>", comboAction)
27 root.mainloop()

```

7.10.5 Usage de ttk.TreeView

L'objet **Treeview** du module **ttk**, permet d'organiser et de visualiser des données sur une **fenêtre Tkinter**. Le rôle de l'objet **Treeview** est similaire à celui de l'objet **JTable** de **Java Swing**. Si vous avez déjà programmer des application graphique GUI Java Swing, vous allez apprendre avec aisance à utiliser ce truc.

Pour bien comprendre et apprécier le fonctionnement de l'objet **Treeview**, nous allons le traiter sur un exemple concret : on souhaite par exemple visualiser les données d'une table sql contenant les données des étudiants :

id	nom	email	age
1	Albert	albert@gmail....	27.0
2	Majid	majid@gmail....	33.0
3	Natalie	natalie@gmai...	21.0
4	David	david@gmail....	24.0
5	Julie	julie@gmail.c...	23.0

Etape 1 : on import le module ttk et on crée une fenêtre tkinter :

```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3 from tkinter import ttk
4 root = Tk()
5 root.geometry("500x200")
6
7 root.mainloop()
```

Etape 2 : on crée l'objet Treeview tout en indiquant les identifiants des colonnes :

```
1 tree = ttk.Treeview(root, columns = (1,2,3,4), height = 5,
2                       show = "headings")
3 tree.place(x=50,y=50, width=400)
```

Etape 3 : on définit l'entête de l'objet Treeview :

```
1 tree.heading(1, text="ID")
2 tree.heading(2, text="Nom")
3 tree.heading(3, text="Email")
4 tree.heading(4, text="Age")
```

Etape 4 : on insère les données :

```
1 tree.insert('', 'end', values = (1, "Albert", "
2     albert@gmail.com",27) )
3 tree.insert('', 'end', values = (2, "Majid", "majid@gmail.
4     com",33) )
5 tree.insert('', 'end', values = (3, "Natalie", "
6     natalie@gmail.com",21) )
7 #... on peut insérer autant de données qu'on souhaite !
```

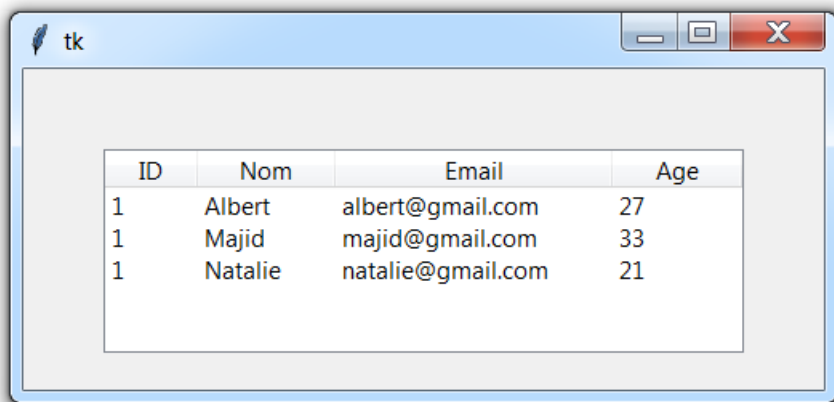
On peut aussi définir les dimensions des colonnes :

```
1 # == dimension des colonnes ==
2 tree.column(1, width = 20)
3 tree.column(2, width = 70)
4 tree.column(3, width = 150)
5 tree.column(4,width=50)
```

Code final :

```
1 from tkinter import *
2 from tkinter import ttk
3 root = Tk()
4 root.geometry("500x200")
5
6 # == Création de l'objet Treeview ==
7 tree = ttk.Treeview(root, columns = (1,2,3,4), height = 5,
8     show = "headings")
9 tree.place(x=50,y=50, width=400)
10
11 # == dimension des colonnes ==
12 tree.column(1, width = 20)
13 tree.column(2, width = 70)
14 tree.column(3, width = 150)
15 tree.column(4,width=50)
16
17 # == Création de l'entête ==
18 tree.heading(1, text="ID")
19 tree.heading(2, text="Nom")
20 tree.heading(3, text="Email")
21 tree.heading(4, text="Age")
22
23 # == Insertion des données ==
24 tree.insert('', 'end', values = (1, "Albert", "
25     albert@gmail.com",27) )
26 tree.insert('', 'end', values = (2, "Majid", "majid@gmail.
27     com",33) )
28 tree.insert('', 'end', values = (3, "Natalie", "
29     natalie@gmail.com",21) )
30
31 root.mainloop()
```

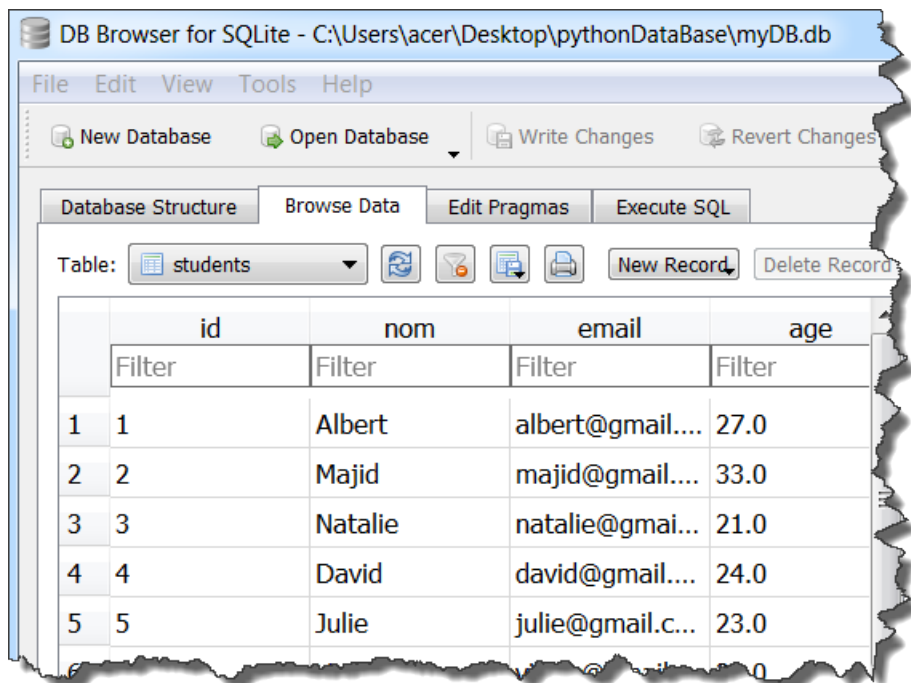
Ce qui affiche après exécution :



ID	Nom	Email	Age
1	Albert	albert@gmail.com	27
1	Majid	majid@gmail.com	33
1	Natalie	natalie@gmail.com	21

7.10.6 Exemple d'utilisation de l'objet Treeview avec une base de données SQLite3

Nous allons dans ce paragraphe reprendre l'exemple de la table **Students** de la base de donnée **mabase SQLite3** du chapitre précédent :



Nous allons essayer de créer un programme Python qui permet à la fois de faire :

1. **une connexion** à la base de donnée **mabase.db**,
2. **extraire les données** depuis la **table Students**
3. **afficher les résultats** dans un objet **Treeview**

```

1  # -*- coding : utf-8 -*-
2  from tkinter import *
3  from tkinter import ttk
4  import sqlite3
5
6  root = Tk()
7  root.geometry("500x200")
8
9  # == Création de l'objet Treeview ==
10 tree = ttk.Treeview(root, columns = (1,2,3,4), height = 5,
11                      show = "headings")
12 tree.place(x=50,y=50, width=400)
13
14 # == dimension des colonnes ==
15 tree.column(1, width = 20)
16 tree.column(2, width = 70)

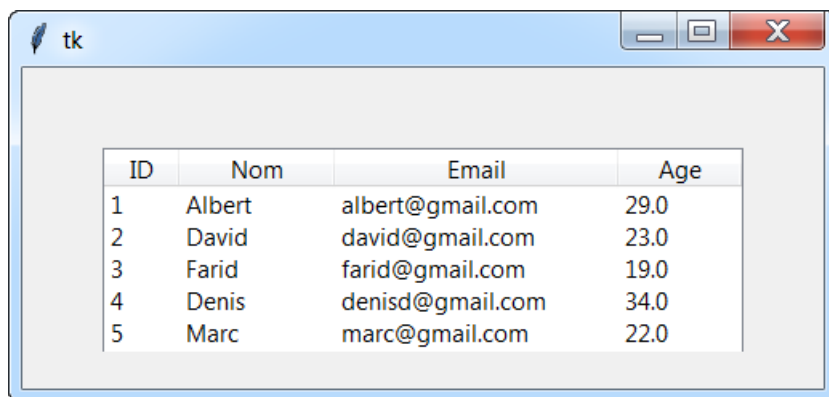
```

```

16 tree.column(3, width = 150)
17 tree.column(4,width=50)
18
19 # == Création de l'entête ==
20 tree.heading(1, text="ID")
21 tree.heading(2, text="Nom")
22 tree.heading(3, text="Email")
23 tree.heading(4, text="Age")
24
25 # == Connexion à la base de données ==
26 conn = sqlite3.connect('mabase.db')
27
28 # == Création d'un cursor et sélection des données ==
29 cur = conn.cursor()
30 result = cur.execute("select * from students ")
31 conn.commit()
32
33 # Insertion des données au sein de l'objet Treeview
34 for row in result :
35     tree.insert('', 'end', values =(row[0], row[1], row
36         [2],row[3]) )
37 conn.close()
38 root.mainloop()

```

Ce qui affiche après exécution :



ID	Nom	Email	Age
1	Albert	albert@gmail.com	29.0
2	David	david@gmail.com	23.0
3	Farid	farid@gmail.com	19.0
4	Denis	denisd@gmail.com	34.0
5	Marc	marc@gmail.com	22.0

7.10.7 Amélioration de l'affichage par une barre de défilement (scrollbar)

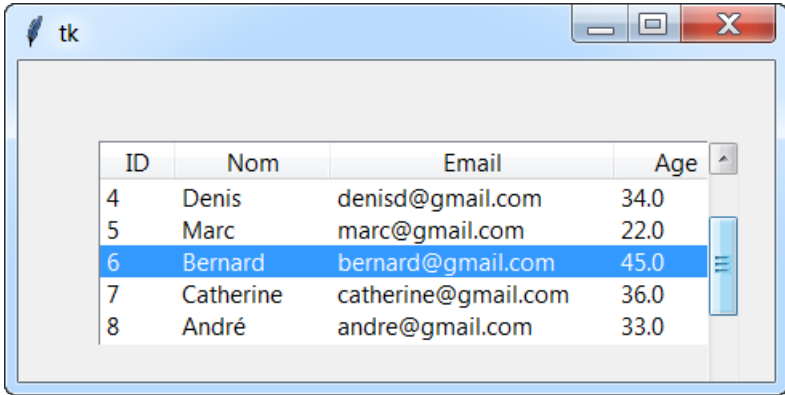
Afin d'améliorer l'affichage des données de l'objet **Treeview**, on peut associer à ce dernier une barre de défilement en ajoutant le code suivant juste après la ligne n°11 :

```

1 scroll = ttk.Scrollbar(root, orient="vertical", command=
    tree.yview)
2 scroll.place(x=430, y=50, width=20, height=200)
3 tree.configure(yscrollcommand=scroll.set)

```

Ajuster les coordonnées d'emplacement **x** et **y** afin de positionner correctement la barre de défilement :



7.11 Obtenir des informations sur la ligne sélectionné

On souhaite parfois obtenir des informations sur une **ligne sélectionnée** de l'objet **Treeview** afin de pouvoir **supprimer** la ligne ou la **mettre à jour** au niveau de la base de donnée... Pour ce faire, on utilise le code suivant qui renvoie la sélection sous forme d'une liste :

```

1 tree.item(tree.selection())['values']

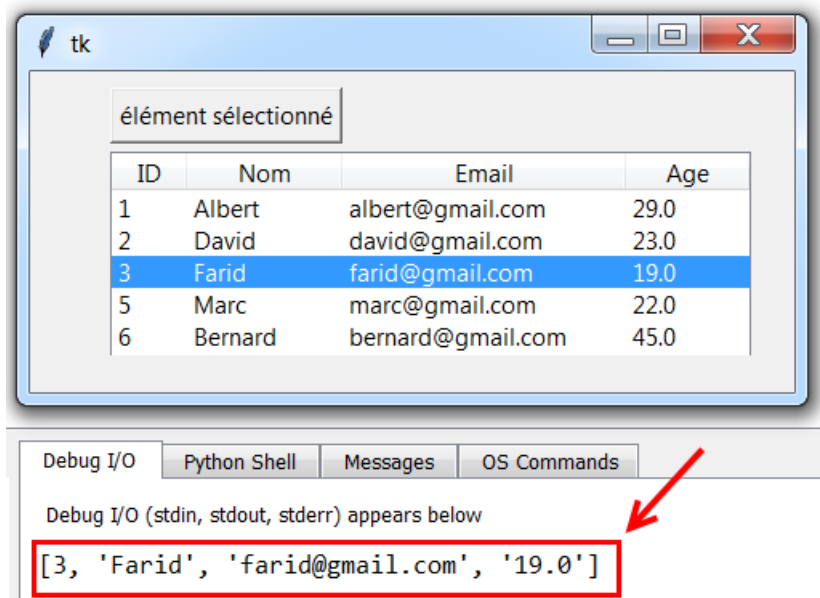
```

Exemple. En ajoutant avant la dernière ligne le code :

```

1 def selectedElement(event):
2     select = tree.item(tree.selection())['values']
3     print(select)
4     Bselect = Button(root, text="élément sélectionné")
5     Bselect.place(x=50, y=10)
6     Bselect.bind("<Button-1>", selectedElement)
7     # Affiche à titre d'exemple en sélectionnant la 3 ème
    ligne : [3, 'Farid', 'farid@gmail.com', '19.0']

```



Exemple. Supprimer un élément sélectionné

```

1 def deleteSelectedElement(event):
2     # Obtention de l'identifiant de l'élément sélectionné
3     id = tree.item(tree.selection())['values'][0]
4     conn = sqlite3.connect('mabase.db')
5     cur = conn.cursor()
6     cur.execute("delete from students where id = {}".format(id))
7     conn.commit()
8     conn.close()
9     Bselect = Button(root, text="Supprimer l'élément
    sélectionné")
10    Bselect.place(x=50, y=10)
11    Bselect.bind("<Button-1>", deleteSelectedElement)
12    # En sélectionnant une ligne et en cliquant sur le bouton '
    Bselect' la ligne sera supprimer
13    # Pour s'assurer que la ligne a bien été supprimée,
    redemarrer l'application

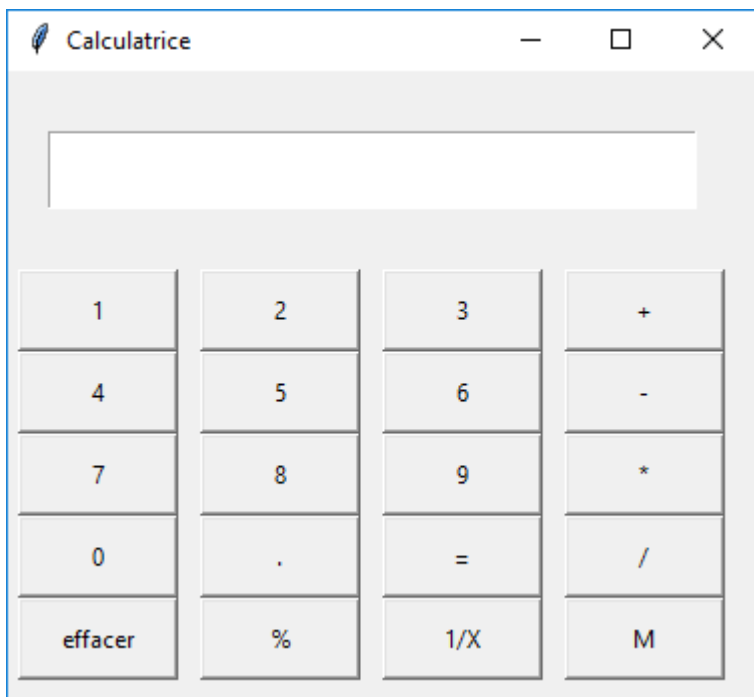
```


Chapitre 8

Minis Projets En PythonTkinter

8.1 Mini Projet : Calculatrice En Python Tkinter

On se propose dans ce **mini projet graphique** en **Python Tkinter** de créer une **simple calculatrice** comme le montre la figure ci-dessous :



8.1.1 Les constituants du projet

1. L'interface graphique sera basée sur la disposition géométrique **grid** (les boutons seront nommés **btn_0**, **btn_1**, **btn_3**, ... , **btn_9**).
2. Le champ de saisie **Entry** sera nommé **equation**.
3. Nous allons ensuite créer une **variable globale** qui sera nommée **formule** qui sera affectée au champ de saisie **equation**.
4. Nous créons ensuite une **méthode** nommée **click()** qui permettra de saisir l'**etiquette** du **bouton** sur lequel le click a été effectué.
5. Nous créons ensuite une **méthode** nommée **equalclick()** qui va **évaluer** l'**expression saisie** et afficher le résultat sur le **champ equation**.
6. Finalement une **méthode effacer()** pour effacer le résultat.

8.1.2 Code de l'application calculatrice

```

1  from tkinter import *
2
3  formule = ""
4
5  def click(num) :
6
7      global formule
8      formule = formule + str(num)
9      equation.set(formule)
10
11 def equalclick() :
12     try :
13         global formule
14
15         result = str(eval(formule))
16         equation.set(result)
17         formule = result
18
19     except :
20         equation.set(" error ")
21         formule = ""
22
23 def effacer() :
24     global formule
25     formule = ""
26     equation.set("")
27
28 if __name__ == "__main__" :
29     master = Tk()
30     master.title("Calculatrice")
31     master.geometry("375x315")
32     equation = StringVar()
33     formule_field = Entry(master, textvariable=equation)
34     formule_field.grid(columnspan=4, pady= 30 , padx = 20
35                        , ipadx = 100 , ipady = 10)
36     btn_1 = Button(master, text=' 1 ', command=lambda :
37                    click(1), height=2, width=10)
38     btn_1.grid(row=2, column=0)
39
40     btn_2 = Button(master, text=' 2 ', command=lambda :
41                    click(2), height=2, width=10)
42     btn_2.grid(row=2, column=1)
43
44     btn_3 = Button(master, text=' 3 ', command=lambda :
45                    click(3), height=2, width=10)
46     btn_3.grid(row=2, column=2)
47
48     btn_4 = Button(master, text=' 4 ', command=lambda :
49                    click(4), height=2, width=10)
50     btn_4.grid(row=2, column=3)

```

```

46
47 btn_5 = Button(master, text=' 5 ', command=lambda :
48 click(5), height=2, width=10)
49 btn_5.grid(row=3, column=1)
50
51 btn_6 = Button(master, text=' 6 ', command=lambda :
52 click(6), height=2, width=10)
53 btn_6.grid(row=3, column=2)
54
55 btn_7 = Button(master, text=' 7 ', command=lambda :
56 click(7), height=2, width=10)
57 btn_7.grid(row=4, column=0)
58
59 btn_8 = Button(master, text=' 8 ', command=lambda :
60 click(8), height=2, width=10)
61 btn_8.grid(row=4, column=1)
62
63 btn_9 = Button(master, text=' 9 ', command=lambda :
64 click(9), height=2, width=10)
65 btn_9.grid(row=4, column=2)
66
67 btn_0 = Button(master, text=' 0 ', command=lambda :
68 click(0), height=2, width=10)
69 btn_0.grid(row=5, column=0)
70
71 plus = Button(master, text=' + ', command=lambda :
72 click("+"), height=2, width=10)
73 plus.grid(row=2, column=3)
74
75 minus = Button(master, text=' - ', command=lambda :
76 click("-"), height=2, width=10)
77 minus.grid(row=3, column=3)
78
79 multiply = Button(master, text=' * ', command=lambda :
80 click("*"), height=2, width=10)
81 multiply.grid(row=4, column=3)
82
83 divide = Button(master, text=' / ', command=lambda :
84 click("/"), height=2, width=10)
85 divide.grid(row=5, column=3)
86
87 equal = Button(master, text=' = ', command=equalclick,
88 height=2, width=10)
89 equal.grid(row=5, column=2)
90
91 effacer = Button(master, text=' effacer ', command=
92 effacer, height=2, width=10)
93 effacer.grid(row=6, column='0')
94
95 Decimal= Button(master, text=' . ', command=lambda :
96 click("."), height=2, width=10)
97 Decimal.grid(row=5, column=1)

```

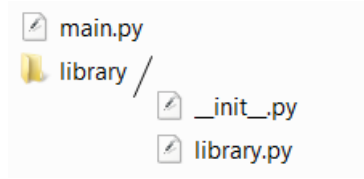
```
85     percent= Button(master, text='%', command=lambda :  
86         click('%'), height=2, width=10)  
87     percent.grid(row=6, column=1)  
88  
89     inverse= Button(master, text='1/X', height=2, width  
90         =10)  
91     inverse.grid(row=6, column=2)  
92  
93     memo= Button(master, text='M', height=2, width=10)  
94     memo.grid(row=6, column=3)  
95  
96     master.mainloop()
```

8.2 Mini projet : création d'un éditeur de texte



8.2.1 Création de l'arborescence du système

Avant de commencer à coder, vous devez faire une conception de votre système. Nous vous proposons la suivante :



1. **library** : représente le répertoire des librairies du système (modules et classes). Ce répertoire contient un fichier vide nommé `__init__.py` qui indique à l'interpréteur de Python qu'il s'agit du répertoire d'un module (ce fichier n'a pas besoin d'être créé, mais il le sera automatiquement).
2. **library.py** : représente la bibliothèque du système qui va contenir toutes les classes Python nécessaire au fonctionnement du système.
3. **main.py** : représente le fichier principal avec lequel démarre le système. Ce fichier ne contient que très peu de code, généralement les instances des classes qui se trouvent sur la bibliothèque **library.py**

8.2.2 Code du fichier library.py

8.2.2.1 Importation de la bibliothèque tkinter

L'éditeur que nous souhaitons créer, manipule des objets graphique : **fenêtres, boutons, labels, boîte de dialogue...** Pour cela nous devons importer une bibliothèque graphique. Nous utilisons la bibliothèque **Tkinter** pour ce projet :

```
1 from tkinter import *
2 import os
3 savedFile = {1: ""}
```

Remarque. la troisième ligne nous permettra de stocker les objets files qui ne tiennent pas dans une instance de classe.

8.2.2.2 Création de la classe principale du fichier library.py

```
1 from tkinter import *
2 import os
3 from tkinter import filedialog
4 savedFile = {1: ""}
5
6 class Win :
7     def __init__(self , master , content ) :
```

```

8         # Fenêtre principale
9         self.master = master
10        # Main Text Widget
11        self.content=content
12
13        # Création de la fenêtre tkinter
14        def create(self) :
15            self.master = Tk()
16            self.master.title("Editeur de Texte")
17            self.master.geometry("700x550")
18
19        # Méthode qui ajoute la zone de texte
20        def add_text(self) :
21            self.content = Text(self.master)
22            self.content.pack(expand=1, fill='both')
23
24        # Génération de la fenêtre principale
25        def generate(self) :
26            self.master.mainloop()

```

8.2.2.3 Ajout des menus

Pour bien comprendre la méthode d'ajout des menus, veuillez réviser le tutoriel sur les menus que nous venons de traiter sur le présent chapitre. Nous passons directement au code qui permet d'ajouter une méthode **add_menu()** qui permet de jouer l'affaire :

```

1    def add_menu(self) :
2        # 1 - Création de la barre des menus
3        menuBar = Menu(self.master)
4        # 2 - Création du menu Fichier
5        menuFichier = Menu(menuBar, tearoff=0)
6        menuBar.add_cascade(label = "Fichier", menu=
menuFichier)
7        # Création des sous menus du menu Fichier
8        menuFichier.add_command(label="Nouveau", command=
self.nouveau)
9        menuFichier.add_command(label="Ouvrir", command=
self.fopen)
10       menuFichier.add_command(label="Enregistrer",
command=self.save)
11       menuFichier.add_command(label="Enregistrer sous",
command=self.saveAs)
12       menuFichier.add_command(label="Quitter", command =
self.quitter)
13
14       #3 - Création du Menu Edition
15       menuEdition= Menu(menuBar, tearoff=0)
16       menuBar.add_cascade(label = "Edition ", menu=
menuEdition)

```

```

17     menuEdition.add_command(label="Annuler")
18     menuEdition.add_command(label="Rétablir")
19     menuEdition.add_command(label="Copier", command=
self.copy)
20     menuEdition.add_command(label="Couper", command =
self.cut)
21     menuEdition.add_command(label="Coller", command=
self.past)
22     #4 – Création du Menu Outils
23     menuOutils = Menu(menuBar,tearoff=0)
24     menuBar.add_cascade(label = "Outils", menu =
menuOutils)
25     menuOutils.add_command(label="Préférences")
26     # Création du Menu Aide
27     menuAide = Menu(menuBar,tearoff=0)
28     menuBar.add_cascade(label = "Aide", menu =
menuAide)
29     menuAide.add_command(label="A propos")
30
31     self.master.config(menu = menuBar)

```

8.2.2.4 Ajout de commandes pour les menus

Jusqu'à présent les menus créés ne sont pas fonctionnels, puisque nous ne leur avons pas encore créé des commandes ! Nous devons donc associer à chaque menu une commande :

```
1 command = nom_de_la_commande
```

Nous devons ensuite créer une **action** pour chaque **commande** et qui porte le même nom :

```

1 def nom_de_la_commande() :
2     #code de l'action ici

```

Code finale du fichier library.py

```

1 # coding : utf-8
2 #-----
3 # Author       : Y. DERFOUFI
4 # Author Title : Professeur Agrégé de mathématiques &
Docteur en Math-Informatiques
5 # Compagny    : CRMEF OUJDA
6 #-----
7 from tkinter import *
8 import os
9 from tkinter import filedialog
10 savedFile = {1:""}
11
12 #-----

```



```

13 # 1 – Classe de la fenêtre pricipale
14 #=====
15 class Win :
16     def __init__(self ,master ,content) :
17         # Fenêtre principale
18         self.master = master
19         # Main Text Widget
20         self.content=content
21     # Création de la fenêtre tkinter
22     def create(self) :
23         self.master = Tk()
24         self.master.title("Editeur de Texte")
25         self.master.geometry("700x550")
26     # Méthode qui ajoute la zone de texte
27     def add_text(self) :
28         self.content = Text(self.master)
29         self.content.pack(expand=1,fill='both')
30     # Génération de la fenêtre principale
31     def generate(self) :
32         self.master.mainloop()
33
34     #=====
35     # 2 – Définition des actions des menus
36     #=====
37     #
38     # 2.1 – actions du menu Fichier
39     #
40     def quitter(self) :
41         self.master.quit()
42     def nouveau(self) :
43         os.popen("python main.py")
44     def fopen(self) :
45         file = self.master.filename = filedialog.
askopenfilename(initialdir = "/",title = "Select File"
,filetypes = (("Text Files","*.txt"),("all files","*.*
")))
46         fp = open(file,"r")
47         r = fp.read()
48         self.content.insert(1.0,r)
49         # Menu Enregistrer sous
50     def saveAs(self) :
51         # create save dialog
52         fichier=self.master.filename=filedialog.
asksaveasfilename(initialdir = "/",title = "
Enregistrer Sous\
",filetypes = (("Fichier
Texte","*.txt"), ("Tous les fichiers","*.*")))
53         fichier = fichier + ".txt"
54         # Utilisation du dictionnaire pour stocker le
fichier
55         savedFile[1] = fichier
56         f = open(fichier,"w")
57         s = self.content.get("1.0",END)

```

```

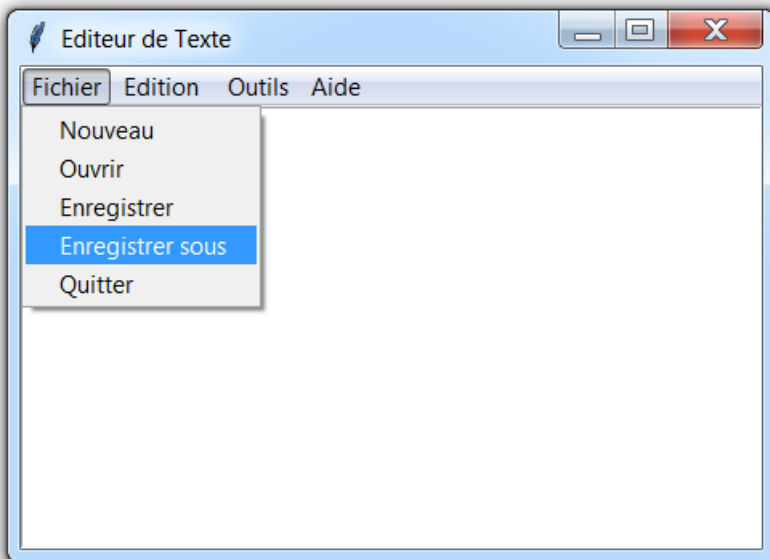
58         f.write(s)
59         f.close()
60         # menu Enregistrer
61     def save(self) :
62         if(savedFile[1] == "") :
63             self.saveAs()
64         else :
65             f = open(savedFile[1], "w")
66             s = self.content.get("1.0",END)
67             f.write(s)
68             f.close()
69
70     # 2.2 – actions du menu Edition
71
72     def copy(self) :
73         self.content.clipboard_clear()
74         self.content.clipboard_append(self.content.
selection_get())
75     def past(self) :
76         self.content.insert(INSERT, self.content.
clipboard_get())
77     def cut(self) :
78         self.copy()
79         self.content.delete("sel.first", "sel.last")
80
81
82     # 2 – Méthodes d'ajout des menus
83
84     def add_menu(self) :
85         # 1 – Création de la barre des menus
86         menuBar = Menu(self.master)
87         # 2 – Création du menu Fichier
88         menuFichier = Menu(menuBar, tearoff=0)
89         menuBar.add_cascade(label = "Fichier", menu=
menuFichier)
90         # Création des sous menus du menu Fichier
91         menuFichier.add_command(label="Nouveau", command=
self.nouveau)
92         menuFichier.add_command(label="Ouvrir", command=
self.fopen)
93         menuFichier.add_command(label="Enregistrer",
command=self.save)
94         menuFichier.add_command(label="Enregistrer sous",
command=self.saveAs)
95         menuFichier.add_command(label="Quitter", command =
self.quitter)
96
97         #3 – Création du Menu Edition
98         menuEdition= Menu(menuBar, tearoff=0)
99         menuBar.add_cascade(label = "Edition ", menu=
menuEdition)
100        menuEdition.add_command(label="Annuler")

```

```
101         menuEdition.add_command(label="Rétablir")
102         menuEdition.add_command(label="Copier", command=
self.copy)
103         menuEdition.add_command(label="Couper", command =
self.cut)
104         menuEdition.add_command(label="Coller", command=
self.past)
105         #4 - Création du Menu Outils
106         menuOutils = Menu(menuBar, tearoff=0)
107         menuBar.add_cascade(label = "Outils", menu =
menuOutils)
108         menuOutils.add_command(label="Préférences")
109         # Création du Menu Aide
110         menuAide = Menu(menuBar, tearoff=0)
111         menuBar.add_cascade(label = "Aide", menu =
menuAide)
112         menuAide.add_command(label="A propos")
113         self.master.config(menu = menuBar)
```

8.2.3 Aperçu de l'éditeur

Si vous avez bien codé votre système et que tout va bien, voici l'aperçu de votre petit éditeur de texte :



8.3 Mini Projet : logiciel de traduction

Nous allons nous baser maintenant sur le module **deep-translator** qui est un outil flexible gratuit et illimité pour traduire entre différentes langues de manière simple en utilisant plusieurs traducteurs. Ce module est disponible sur la plate-forme **pypi** :

<https://pypi.org/project/deep-translator/>.

8.3.1 Installation du module deep-translator

L'installation module **deep-translator** se fait avec aisance avec la commande **cmd** via l'utilitaire **pip** :

```
1 pip install -U deep-translator
```

8.3.2 Syntaxe

Avant de traduire un texte, on commence par importer le module :

```
1 from deep_translator import GoogleTranslator
```

On utilise ensuite la classe **GoogleTranslator** pour traduire un texte :

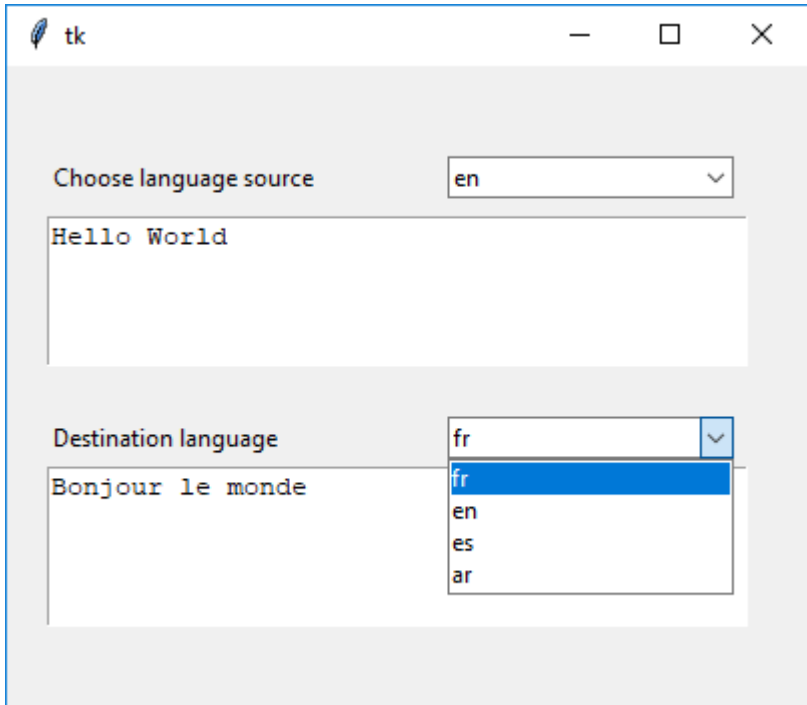
```
1 texte_traduit = GoogleTranslator(source='langue_source',  
    target='langue_destination').translate("texte à  
    traduire")
```

Exemple. (Traduire un texte anglais en français)

```
1 # coding : utf-8  
2 from deep_translator import GoogleTranslator  
3 translated = GoogleTranslator(source='en', target='fr').  
    translate("I start up learning Python")  
4 print(translated) # affiche : Je commence à apprendre  
    Python
```

8.3.3 Mini logiciel de traduction avec tkinter et deep-translator

Nous allons voir maintenant comment peut améliorer cette application de traduction à l'aide de la bibliothèque **tkinter** :



On commence par importer les modules nécessaires :

```
1 from tkinter import *
2 from tkinter import ttk
3 from deep_translator import GoogleTranslator
```

On définit ensuite le texte à traduire, les langues source et destination comme variables globales :

```
1 #définir les langues source et destination sous forme de
   variables globales
2 source = "fr"
3 destination = "en"
4 # texte à traduire comme variable globale
5 t = ""
```

On crée ensuite l'interface graphique :

```
1 from tkinter import *
2 from tkinter import ttk
3 from deep_translator import GoogleTranslator
4
5 #définir les langues source et destination sous forme de
   variables globales
```

```

6 source = "fr"
7 destination = "en"
8 # texte à traduire comme variable globale
9 t = ""
10
11 root = Tk()
12 root.geometry("400x320")
13
14 # -----
15 # Création de la liste combobox
16 # -----
17 labelChooseLang = Label(root, text = "Choose language
    source")
18 labelChooseLang.place(x = 20 , y = 45)
19
20 labelLangTraduct = Label(root, text = "Destination
    language")
21 labelLangTraduct.place(x = 20 , y = 175)
22
23
24 # Liste des valeurs d'option de la combobox
25 languages =['fr' , 'en' , 'es' , 'ar']
26
27 # Création des listes combobox
28 combo1 = ttk.Combobox(root, values = languages )
29 combo1.place(x = 220 , y = 45)
30 # Définir l'élément qui s'affiche par défaut
31 combo1.current(0)
32
33 combo2 = ttk.Combobox(root, values = languages)
34 combo2.place(x = 220 , y = 175)
35 # Définir l'élément qui s'affiche par défaut
36 combo2.current(1)
37
38 T1 = Text(root )
39 T1.place(x = 20 , y = 75 , width = 350 , height = 75)
40
41 T2 = Text(root)
42 T2.place(x = 20 , y = 200 , width = 350 , height = 80)
43
44 root.mainloop()

```

On ajoute ensuite les **actions** du type **event** aux listes des langues **combobox**, qui auront pour rôle, la récupération de la langue sélectionnée :

```

1 combo1.bind("<<ComboboxSelected>>", comboAction)
2 combo2.bind("<<ComboboxSelected>>", comboAction)

```

Et sans oublier de créer la méthode qui réalisent les actions :

```

1 def comboAction(event) :

```

```

2     global source
3     global destination
4     source = combo1.get()
5     destination = combo2.get()

```

On associe ensuite une **action** du type **event** au **premier champ** de saisie qui sera déclenché après avoir saisi le texte source, dès que appui sur la **touche entrée** du clavier :

```

1 T1.bind("<Return>" , Traduct)

```

Et finalement la méthode **Traduct** qui réalise l'action :

```

1 def Traduct(event) :
2     global t
3     t = ""
4     t = T1.get("1.0" , END)
5
6     # traduction du texte
7     translated = GoogleTranslator(source = source , target=
8     destination).translate(t)
9     T2.delete('1.0' , END)
10    T2.insert(END , translated)

```

Code final de l'application :

```

1  # coding : utf-8
2  """
3  # Younes Derfoufi
4  # chanel : https://www.youtube.com/user/
5  InformatiquesFacile
6  # Site : https://www.tresfacile.net/
7
8  # Vous devez installer le module de traduction deep-
9  translator
10
11  # pip install -U deep-translator
12  """
13
14  from tkinter import *
15  from tkinter import ttk
16  from deep_translator import GoogleTranslator
17
18  #définir les langues source et destination sous forme de
19  variables globales
20  source = "fr"
21  destination = "en"
22  # texte à traduire comme variable globale
23  t = ""
24
25  def comboAction(event) :
26      global source

```

```

24     global destination
25     source = combo1.get()
26     destination = combo2.get()
27
28
29 def Traduct(event) :
30     global t
31     t = ""
32     t = T1.get("1.0" , END)
33
34     # traduction du texte
35     translated = GoogleTranslator(source = source , target=
36     destination).translate(t)
37     T2.delete('1.0' , END)
38     T2.insert(END , translated)
39
40
41 root = Tk()
42 root.geometry("400x320")
43
44 #-----
45 # Création de la liste combobox
46 #-----
47 labelChooseLang = Label(root , text = "Choose language
48     source")
49 labelChooseLang.place(x = 20 , y = 45)
50
51 labelLangTraduct = Label(root , text = "Destination
52     language")
53 labelLangTraduct.place(x = 20 , y = 175)
54
55 # Liste des valeurs d'option de la combobox
56 languages =['fr' , 'en' , 'es' , 'ar']
57
58 # Création des listes combobox
59 combo1 = ttk.Combobox(root , values = languages )
60 combo1.place(x = 220 , y = 45)
61 # Définir l'élément qui s'affiche par défaut
62 combo1.current(0)
63 # Associé une bind action à la liste combo
64 combo1.bind("<<ComboboxSelected>>" , comboAction)
65
66 combo2 = ttk.Combobox(root , values = languages)
67 combo2.place(x = 220 , y = 175)
68 # Définir l'élément qui s'affiche par défaut
69 combo2.current(1)
70 # Associé une bind action à la liste combo
71 combo2.bind("<<ComboboxSelected>>" , comboAction)
72
73 T1 = Text(root )

```



```
73 T1.place(x = 20 , y = 75 , width = 350 , height = 75)
74 T1.bind("<Return>" , Traduct)
75
76 T2 = Text(root)
77 T2.place(x = 20 , y = 200 , width = 350 , height = 80)
78
79 root.mainloop()
```

8.3.4 Code source

Le code source de ce projet est disponible sur **Pastebin** : <https://pastebin.com/GBNGS5ZF>

Troisième partie

Exercices Avec Solutions

1. Exercices sur les bases en python : strings & variables

Exercice 1. Écrire un programme en langage Python qui demande à l'utilisateur de saisir son nom et de lui afficher son nom avec un message de bienvenue!

Solution.

```
1 nom = input("Tapez votre nom : ")
2 print("Bienvenue : " , nom)
```

Exercice 2. Écrire un programme en Python qui demande à l'utilisateur de saisir deux nombres **a** et **b** et de lui afficher leur **somme** : **a + b**.

Solution.

```
1 #coding : utf-8
2 # demander à l'utilisateur de saisir les valeurs de a et
  de b
3 a = input("Tapez la valeur du nombre a : ")
4 b = input("Tapez la valeur du nombre b : ")
5 # Convertir les chaînes de caractères en entier
6 a = int(a)
7 b = int(b)
8 s = a+b
9 print("La somme de a et de b est a + b = " , s)
```

Exercice 3. Écrire un programme en langage Python qui demande à l'utilisateur de saisir **deux nombres a et b** et de lui afficher leur **maximum**, sans utiliser la fonction **max()** ni aucune fonction prédéfinie.

Solution.

```

1 # coding : utf-8
2 # lire les valeurs de a et b
3 a = int(input("Tapez la valeur du nombre a : "))
4 b = int(input("Tapez la valeur du nombre b : "))
5
6 # Faire un test de comparaison pour trouver le plus grand
7 if (a > b) :
8     print("Le maximum de a et de b est : a = ", a)
9 else :
10    print("Le maximum de a et de b est : b = ", b)

```

Exercice 4. Écrire un programme en langage Python qui affiche les 100 premiers nombres entiers : 1 , 2 , ... , 100

Solution.

```

1 # coding : utf-8
2 # parcourir les 100 premiers nombres à l'aide de la boucle
  for
3 for i in range(1,101) :
4     print(i)
5 """
  après exécution le programme affiche les 100 premiers
  nombres :
6 1
7 2
8 .
9 .
10 .
11 100

```

Exercice 5. Écrire un programme en langage Python qui demande à l'utilisateur de saisir un **nombre entier** et de lui afficher si ce nombre est **pair** ou **impair**.

Solution.

```

1 # coding : utf-8
2 # Lire la valeur de n

```

```

3 n = input("Type value of the integer n : ")
4
5 # Convertir n en entier
6 n = int(n)
7 # Tester si n est pair ou non
8 if (n%2 == 0) :
9     print("Le nombre '", n, "' tapé est pair ")
10 else :
11     print("Le nombre '", n, "' tapé est impair ")

```

Exercice 6. Écrire un programme en langage Python qui demande à l'utilisateur de saisir son âge et de lui afficher le message '**vous êtes Majeur !**' si l'âge tapé est **supérieur ou égale à 18** et le message '**vous êtes mineur !**' si l'âge tapé est **inférieur à 18**.

Solution.

```

1 # coding : utf-8
2 # Demander à l'utilisateur de taper son âge
3 age = int(input("Tapez votre age : "))
4 if (age > 18) :
5     print("Vous êtes majeur !")
6 else :
7     print("Vous êtes mineur !")

```

Exercice 7. Écrire un programme en Python qui demande à l'utilisateur de saisir **3 nombre x, y et z** et de lui afficher leur **maximum** sans utiliser aucune fonction prédéfinie.

Solution.

```

1 # coding : utf-8
2 # Demander à l'utilisateur de taper 3 nombres a, b, c
3 a = int(input("Type a value of the number a "))
4 b = int(input("Type a value of the number b "))
5 c = int(input("Type a value of the number c "))
6
7 # définir et initialiser le maximum à zero
8 max = 0
9 if (a > b) :
10     max = a
11 else :
12     max = b
13 if (max < c) :

```

```

14     max = c
15 else :
16     max = max
17 print("Le maximum des trois nombre est : max(a,b,c) = ",
      max)

```

Exercice 8. Écrire un programme en Python qui demande à l'utilisateur de saisir un **nombre entier n** et de lui afficher la valeur de la somme $s = 1 + 2 + \dots + n = ?$

Solution.

```

1 # coding : utf-8
2 # Demander à l'utilisateur de saisir la valeur de l'entier
  n
3 n = int(input("Type a value of the integer n "))
4
5 # define et initialiser une variable auxiliaire j
6 j = 0
7 for i in range(1,n+1) :
8     j = j + i
9 print("La somme 1 + 2 + 3 + ... + ",n," = : ", j)

```

Exercice 9. Écrire un programme en Python qui demande à l'utilisateur de saisir un **nombre entier n** et de lui afficher **n!**

Solution.

```

1 # coding : utf-8
2 # Demander à l'utilisateur de saisir la valeur de l'entier
  n
3 n = int(input("Type a value of the integer n "))
4
5 # initialiser une variable auxiliaire j
6 j = 1
7 for i in range(1,n+1) :
8     j = j*i
9 print("Factorielle de n est : ",n,"! = : ", j)

```

Exercice 10. Écrire un programme en Python qui demande à l'utilisateur de saisir le **rayon r** d'un **cercle** et de lui renvoyer la **surface** et le **périmètre** du cercle.

Solution.

```

1 # coding : utf-8
2 # importer le nombre pi à partir de la bibliothèque math
3 from math import pi
4 # lire la valeur du rayon r
5 r = float(input("Saisissez la valeur du rayon r : "))
6 # calcul du périmètre du cercle
7 P = 2*pi*r
8 # calcul de la surface du cercle
9 S = pi*(r**2)
10 print("Le périmètre du cercle de rayons r =",r, " est P = "
        , P)
11 print("La surface du cercle de rayons r =",r, " est S = ",
        S)

```

Exercice 11. Écrire un programme en Python qui demande à l'utilisateur de saisir un nombre **entier** **n** et de lui afficher tous les **diviseurs** de ce nombre.

Solution.

```

1 # coding : utf-8
2 n = int(input("Tapez la valeur de l'entier n "))
3 # parcourir tous les entiers inférieurs ou égaux à n
4 for i in range(1,n+1):
5     # tester si i est un diviseur de n
6     if (n%i==0):
7         print("Le nombre ",i," est un diviseur de ",n)

```

Exercice 12.

1) – Écrire un programme en Python qui demande à l'utilisateur de saisir un nombre entier **n** et de lui afficher la table de multiplication de ce nombre.

2) – Améliorez le programme afin qu'il affiche les tables de multiplications de tous les nombres compris entre 1 et 9

Solution.**1) - Table de multiplication d'un entier saisi au clavier**

```

1 # coding : utf-8
2 # Lire la valeur de l'entier n
3 n = int(input("Tapez la valeur de n "))
4 print("La table de multiplication de : ", n, " est :")
5 for i in range(1,10):
6     print(i , " x " , n, " = " ,i*n)

```

2) - Tables de multiplications de tous les nombres 1, 2, 3, ..., 9

```

1 # coding : utf-8
2 for n in range(1,10) :
3     #insert separator
4     print("_____")
5     print("la table de multiplication de  :", n," est :")
6     for i in range(1,10) :
7         print(i , " x ", n, " = ", i*n)

```

Exercice 13. Écrire un programme en langage Python qui demande à l'utilisateur de saisir **deux nombres entiers a et b** et de lui afficher le **quotient** et le **reste** de la **division euclidienne** de a par b.

Solution.

```

1 # coding : utf-8
2 # Lire les valeurs de a et b
3 a = int(input("Tapez la valeur de l'entier a  : "))
4 b = int(input("Tapez la valeur de l'entier b  : "))
5 q = a//b
6 r = a%b
7 print("Le quotient de la division euclidienne de a par b
      est : q = ", q)
8 print("Le reste de la division euclidienne de a par b est
      : r = ", r)

```

Exercice 14. Écrire un programme en langage Python qui demande à l'utilisateur de saisir un **nombre entier n** et de lui afficher si ce nombre est **carré parfait** ou non.

Solution.

```

1 # coding : utf-8
2 # Lire la valeur de l'entier n
3 n = int(input("Tapez la valeur de n : "))
4 # On utilise un compteur j
5 j = 0
6 for i in range(0,n) :
7     if(i**2 == n) :
8         j = j +1
9 if(j > 0) :
10     print("L'entier  ", n , " est un carré parfait")

```



```

11 else :
12     print("l'entier ", n , " n'est pas est un carré
    parfait")

```

Exercice 15. Écrire un programme en langage Python qui demande à l'utilisateur de saisir un nombre entier n et de lui afficher si ce nombre est premier ou non.

Solution.

```

1 # coding : utf-8
2 # Lire la valeur de l'entier n
3 n = int(input("Tapez la valeur de n : "))
4 # on utilise un compteur qui compte le nombre de diviseurs
  de n
5 j = 0
6 for i in range(1, n+1):
7     if(n%i == 0):
8         j = j + 1
9
10 # On teste si le nombre de diviseurs de n est = 2 pour
    conclure que n est premier
11 if( j == 2):
12     print("Le nombre ", n , " est premier")
13 else :
14     print("Le nombre ", n , " n'est pas premier")

```

Exercice 16. Écrire un programme en langage Python qui permet de parcourir et afficher les caractères d'une variable du type chaîne de caractères. Exemple pour $s = \text{'Python'}$, le programme affiche les caractères :

P
y
t
h
o
n

Solution.

```

1 # coding : utf-8
2 # Demander à l'utilisateur de saisir une chaîne de
  caractères
3 s = input("Saisir une chaîne de caractères s : ")

```

```

4
5 # obtenir la longueur de s
6 n = len(s)
7 for i in range(0, n) :
8     print(s[i])

```

Exercice 17. Écrire un programme en Python permettant d’afficher pour une chaîne de caractères donnée, le nombre d’occurrences de chaque caractère dans la chaîne.

Exemple pour la chaîne de caractère `s = 'Python.org'` le programme doit afficher :

Le caractère : " P " figure 1 fois dans la chaîne s
 Le caractère : " y " figure 1 fois dans la chaîne s
 Le caractère : " t " figure 1 fois dans la chaîne s
 Le caractère : " h " figure 1 fois dans la chaîne s
 Le caractère : " o " figure 2 fois dans la chaîne s
 Le caractère : " n " figure 1 fois dans la chaîne s
 Le caractère : " . " figure 1 fois dans la chaîne s
 Le caractère : " r " figure 1 fois dans la chaîne s
 Le caractère : " g " figure 1 fois dans la chaîne s

Solution.

```

1
2 # coding : utf-8
3 s = "Python est un langage de programmation"
4 # regrouper les caractères de s dans un ensemble pour
   éviter les répétitions
5 unique = set({})
6 for x in s :
7     if x not in unique :
8         unique.add(x)
9         print("Le nombre d'occurrences du caractère : ", x,
              " dans la chaîne s est :", s.count(x))

```

Exercice 18. Écrire un programme en Python qui demande à l’utilisateur de saisir une **chaîne de caractère s** et de lui renvoyer un message indiquant si la chaîne contient la **lettre 'a'** tout en indiquant sa position sur la chaîne. Exemple si l’utilisateur tape la chaîne `s = 'langage'` , le programme lui renvoie :

La lettre 'a' se trouve à la position : 1

La lettre 'a' se trouve à la position : 4

Solution.

```

1 # coding : utf-8
2 # Demander à l'utilisateur de taper la valeur de s
3 s = input("Tapez la valeur de s : ")
4
5 # Obtenir la longueur de la chaîne s
6 n = len(s)
7
8 # Parcourir la chaîne s tout en recherchant le caractère '
  a'
9 for i in range(0,n) :
10     # Tester si le caractère rencontré est égale à 'a'
11     if(s[i] == 'a') :
12         print("Le caractère 'a' se trouve à la position
          : ", i , " dans la chaîne s")

```

Exercice 19. Écrire un programme en Python qui permet de lister les chaînes qui composent la liste **L** = ["laptop", "iphone", "tablet"] tout en indiquant la longueur de chaque chaîne.

Solution.

```

1 l = ["laptop", "iphone", "tablet", "printer", "Ipad"]
2 # parcourir les éléments de la liste l
3 for x in l :
4     print(x, " est dans la liste L, sa longueur est : ",
          len(x))

```

Exercice 20. Écrire un programme en langage Python, permettant d'échanger le premier et le dernier caractère d'une chaîne donnée.

Solution.

```

1 # coding : utf-8
2 # définir un exemple de chaîne s
3 s = "www.tresfacile.net"
4 # obtenir la longueur de la chaîne s
5 n = len(s)
6 # obtenir le premier caractère de la chaîne s
7 first = s[0]
8 last = s[n-1]
9 # extraire la sous chaîne obtenue à partir de s en
  ignorant le premier et le dernier caractère
10 s1 = s[1:n-1]

```

```

11 # construire une nouvelle chaîne en changeant le premier
    et le dernier caractère
12 s2 = last + s1 + first
13 print(s2)

```

Exercice 21. Écrire un programme en langage Python, qui permet de compter le nombre de voyelles dans une chaîne donnée. Exemple pour la chaîne $s = \text{'anticonstitutionnellement'}$ le programme doit renvoyer le message suivant : La chaîne **'anticonstitutionnellement'** possède 10 voyelles.

Solution.

```

1 # définir les voyelles dans un ensemble
2 vowels= {'a','e','y','u','i','o'}
3 # définir une chaîne de caractères
4 s = "anticonstitutionnellement"
5 # obtenir la longueur de la chaîne s
6 n = len(s)
7 # initialiser le nombre de voyelles à 0
8 number_vowels = 0
9 # parcourir les caractères de la chaîne s
10 for i in range(0,n):
11     if(s[i] in vowels):
12         number_vowels = number_vowels + 1
13 print("Le nombre de voyelles de la chaîne 's' est :",
        number_vowels)

```

Exercice 22. Écrire un programme en Python, qui permet de renvoyer le premier mot d'un texte donné. Exemple pour le texte : **T = 'Python est un merveilleux langage de programmation'**, le programme doit renvoyer le mot **'Python'**

Solution.

```

1 # coding : utf-8
2 # programme qui détermine le premier mot d'une chain de
    cractères en Python
3 # définir une chaîne de caractères s
4 s = "Apprendre Python sur www.tresfacile.net"
5
6 # initialiser le premier mot à une chaîne vide
7 premierMot = ""
8
9 # initialiser le compteur

```

```

10 i = 0
11 # chercher le premier espace sur la chaîne
12 while (s[i] != " "):
13     premierMot = premierMot + s[i]
14     i = i + 1
15 print("Le premier mot de la chaîne s est  :", s[:i])

```

Exercice 23. Écrire un programme en langage Python qui demande à l'utilisateur de saisir le nom d'un fichier et de lui renvoyer son extension. Exemple si l'utilisateur saisie '**coursPython.pdf**', le programme lui renvoie le message « **L'extension du fichier est .pdf** »

Solution.

```

1 # coding: utf-8
2 # Demander à l'utilisateur de saisir un nom de fichier
3 fichier = input("Saisir le nom du fichier avec son
    extension: ")
4 # convertir fichier en liste
5 L = fichier.split(".")
6 extensionFichier = "." + L[-1]
7 print("L'extension du fichier est :", extensionFichier)

```

Exercice 24. Un **palindrome** est un mot dont l'ordre des lettres reste le même si on le lit de gauche à droite ou de droite à gauche. Par exemple : '**laval**', '**radar**', '**sos**'... sont des palindromes. Écrire un programme en Python qui demande à l'utilisateur de saisir un mot et de lui renvoyer s'il s'agit d'un palindrome ou non ?

Solution.

```

1 # coding: utf-8
2 # Demander à l'utilisateur de saisir un mot
3 mot = input("Saisir un mot : ")
4 # inverser le mot
5 inverse = mot[::-1]
6 if(mot == inverse):
7     print("Le mot :", mot, " est un palindrome")
8 else:
9     print("Le mot :", mot, " n'est pas un palindrome")

```

Exercice 25. Écrire un programme qui demande à l'utilisateur de saisir un **mot** et de lui renvoyer son **inverse**. Exemple si l'utilisateur saisi le mot **'python'**, le programme lui renvoie le mot **'nohtyp'**.

Solution.

1^{ère} méthode

```

1 # coding : UTF-8
2
3 # Lire la variable string s
4 s = input("Tapez une chaîne s : ")
5
6 # obtenir l'inverse de la chaîne s
7 s1 = s[::-1]
8
9 print("L'inverse de la chaîne : ", s, "' est : ", s1)
```

2^{ème} méthode

```

1 # coding : UTF-8
2 # Lire la variable string s
3 s = input("Tapez une chaîne s : ")
4 # initialiser l'inverse à une chaîne vide
5 inv = ""
6
7 # construction de l'inverse d'une façon récursive
8 for x in s :
9     inv = x + inv
10 print("L'inverse de la chaîne : ", s, "' est : ", inv)
```

Exercice 26. Écrire un programme qui demande à l'utilisateur de saisir un texte et de lui renvoyer tous les mots commençant par la lettre a.

Solution.

1^{ère} méthode

```

1 # coding : UTF-8
2 # Lire la chaîne s
3 s = input("Tapez une chaîne de caractères s : ")
4 # convertir la chaîne s en une liste
5 s = s.split()
6 # chercher tous les éléments de la liste qui commencent
  par la lettre 'a'
7 for x in s :
8     if(x[0] == 'a') :
9         print("Le mot : ", x, " commence par la lettre 'a'")
10
```

2^{ème} méthode

```

1 # coding : UTF-8
2 # Lire la chaîne s
3 s = input("Tapez une chaîne de caractères s : ")
4
5 # convertir la chaîne s en une liste
6 s = s.split()
7
8 # obtenir la longueur de la liste s
9 n = len(s)
10
11 # rechercher les éléments de la liste qui commencent par
    la lettre 'a'
12 for i in range(0,n) :
13     if(s[i][0] == 'a') :
14         print("Le mot : '", s[i], "' commence par
            la lettre 'a'")
15
16 """
17 Example : si on tape le message "Me and you are beginner"
18 On obtient le message :
19 The word : 'and' begin with the letter 'a'
20 The word : 'are' begin with the letter 'a'
21 """

```

Exercice 27. Écrire un programme en Python sous forme de fonction qui calcul la somme des éléments d'une liste de nombres, et un autre programme qui permet de multiplier tous les éléments d'une liste de nombres.

Solution.

```

1 # coding : UTF-8
2 # Fonction qui calcul le produit des éléments d'une liste
3 def mult(l) :
4     m = 1
5     # Création du produit en parcourant les éléments de la
        liste
6     for x in l :
7         m = m*x
8     return m
9 print(mult([2,5,3]))
10
11 # Fonction qui calcul la somme des éléments d'une liste
12 def sum(l) :
13     s = 0
14     # Création de la somme en parcourant les éléments de
        la liste
15     for x in l :

```

```

16         s = s + x
17     return s
18 print(sum([2,5,3]))

```

Exercice 28. Écrire un programme Python qui teste si une liste est vide ou non. Même question pour une chaîne de caractères.

Solution.

1^{ère} méthode

```

1 # coding : UTF-8
2 # Tester si une liste L est vide
3 # définir une liste L
4 L = list()
5
6 # essayer si la liste L est vide
7 if L == [] :
8     print ("La liste L est vide")
9 else :
10    print ("La liste L n'est pas vide")
11
12 #Tester si une chaîne est vide
13 # définir une chaîne s
14 s = ""
15
16 # Tester si une chaîne s est vide
17 if s == "" :
18     print ("La chaîne s est vide")
19 else :
20     print ("La liste n'est pas vide")

```

2^{ème} méthode

```

1 # Tester si une liste Le est vide
2 # définir une liste L
3 L = list()
4
5 # 1) – Tester si la liste L est vide en utilisant la
   fonction len ()
6 if len(L) == 0 :
7     print ("La liste L est vide")
8 else :
9     print ("La liste L n'est pas vide")
10
11
12 #2) – Tester si une chaîne s est vide
13 # définir une chaîne s
14 s = ""

```



```

15
16 #Tester si la chaîne s est vide en utilisant la fonction
    len ()
17 if len (s) == 0 :
18     print ("La chaîne s est vide")
19 else :
20     print ("La liste L n'est pas vide")

```

Exercice 29. Écrire un programme Python qui permet de supprimer les éléments dupliqués d'une liste.

Solution.

1^{ère} méthode

```

1 # coding : UTF-8
2 # définit une fonction qui supprime les doublons dans la
    liste
3 def removeDuplicate(l) :
4     # définir et initialiser la liste sans élément
        dupliqué
5     unique = []
6     # construction de la liste sans éléments dupliqués
7     for x in l :
8         if x not in unique :
9             unique.append(x)
10    return unique
11
12 # Exemple
13 l = [2, 7, 13, 2, 17, 13, 2, 7, 13]
14 print (removeDuplicate(l))

```

2^{ème} méthode

```

1 # coding : utf-8
2 # définir la fonction qui supprime les éléments dupliqués d'
    une liste
3 def removeDuplicate(l) :
4     # convertir la liste en un ensemble
5     SET = set(l)
6     # reconverter l'ensemble en une liste
7     L = list(SET)
8     return L
9
10 # Exemple
11 l = [2, 7, 7, 13, 2, 17, 25, 17, 13, 15, 15, 2, 7, 13]
12 print(removeDuplicate(l))

```

Exercice 30. Écrire une fonction en Python qui permet de comparer deux listes et de nous indiquer si ces deux listes ont une valeur communes ou non.

Solution.

```

1 # coding : UTF-8
2 def elementsCommun(l1 , l2) :
3     compteur = 0
4     for x in l1 :
5         if x in l2 :
6             compteur =compteur + 1
7     if compteur != 0 :
8         return True
9     else :
10        return False
11
12 l1 = [2,35,5,6,21]
13 l2 = [2,13,5,7,19]
14 print(elementsCommun(l1 , l2))
15 # Ce qui affiche True

```

Exercice 31. Écrire un programme Python qui permet d'extraire la liste des entiers pairs et la liste des entiers impairs d'une liste de nombres.

Solution.

```

1 # coding : UTF-8
2 def extract(l) :
3     pair = []
4     impair = []
5     for x in l :
6         if(x%2 == 0) :
7             pair.append(x)
8         else :
9             impair.append(x)
10    print("La liste des entiers pair est :",pair)
11    print("La liste des entiers impair est :",impair)
12 # Tester l'algorithme
13 l =[23,4,56,7,8,9,0,18,7,6,55,43,2]
14 print(extract(l))
15 """
16 Ce qui affiche après exécution :
17 La liste des entiers pair est :  [4, 56, 8, 0, 18, 6, 2]
18 La liste des entiers impair est :  [23, 7, 9, 7, 55, 43]
19 """

```

Exercice 32. Écrire un programme en Python qui renvoie toutes les listes obtenues en permutant les termes d'une liste donnée.

Solution.

```

1 # coding : UTF-8
2 """
3 importation du module itertools :
4 Ce module implémente de nombreux blocks d'itérateurs
   inspirées des constructions APL, Haskell et SML.
5 """
6 import itertools
7 # Exemple d'utilisation
8 l = [1,2,3,4]
9 permutations = itertools.permutations(l)
10 L = list(permutations)
11 print ("Les listes obtenues en échangeant les termes de la
        liste l :\n", L)

```

Exercice 33. Écrire un programme en Python qui demande à l'utilisateur de saisir une chaîne de caractères et d'afficher les caractères d'indice pair. Exemple pour la chaîne $s = \text{« Python »}$, le programme renvoie « Pto » .

Solution.

1^{ère} méthode

```

1 # coding : utf-8
2 # Lire la chaîne s
3 s = input("Saisissez la chaîne s :")
4 s1 = ""
5 i = 0
6 while(i < len(s)-1):
7     s1 = s1 + s[i]
8     i = i +2
9 print(s1)

```

2^{ème} méthode

```

1 # coding : utf-8
2 """
3 # Lire la chaîne s
4 s = input("Saisissez la chaîne s :")
5
6 # parcourt de la chaîne s avec le pas = 2
7 print(s[0:len(s):2])

```

Exercice 34. Étant donnée la liste des notes des élèves : **notes** = [12 , 04 , 14 , 11 , 18 , 13 , 07, 10 , 05 , 09 , 15 , 08 , 14 , 16]. Écrire un programme Python qui permet d'extraire de cette liste et créer une autre liste qui contient uniquement les notes au dessus de la moyenne (les *notes* ≥ 10)

Solution.

```

1 # coding : utf-8
2 notes = [12, 4, 14, 11, 18, 13, 7, 10, 5, 9, 15, 8, 14,
           16]
3
4 # définir la liste qui contiendra les notes au-dessus de
   la moyenne
5 moyenne = []
6 for x in notes :
7     # ajouter uniquement les notes >= 10 à la liste
       moyenne
8     if(x >= 10) :
9         moyenne.append(x)
10
11 print ("La liste contenant uniquement les notes
        supérieures à la moyenne est :\n", moyenne)

```

Exercice 35. Écrire un programme en Python qui permet de transformer une adresse **url** saisie au clavier en un lien hypertexte.

Solution.

```

1 # coding : utf-8
2 # Lire adresse url
3 url = input("Saisir une url :")
4 # Lire le texte du lien hypertexte
5 text_lien = input("saisir le texte du lien")
6 # convert the url text to a link
7 url = "<a href='"+ url + "'> " + text_lien + "</a>"
8 print(url)

```

Exercice 36. Écrire un programme en Python permettant de supprimer les espaces multiples dans une chaîne s.

Solution.

```

1 # coding : utf-8
2 s = "Exemple      de texte avec des      espaces      multiple
   "
3 # Transformer le texte en une liste
4 L = s.split()
5
6 # initialiser le texte sans espace à un texte vide
7 texteSansEspace = ""
8
9 # Reconstruire le texte en parcourant les éléments de la
   liste
10 for x in L :
11     texteSansEspace = texteSansEspace + x + " "
12 print(texteSansEspace)

```

Exercice 37. Écrire un programme Python qui permet de regrouper dans une liste les **mots communs** à deux **chaînes s1** et **s2**.

Solution.

```

1 # coding : utf-8
2 def motsCommuns(s1,s2) :
3     L1 = s1.split()
4     L2 = s2.split()
5     communs = []
6     for x in L1 :
7         if(x in L2) :
8             communs.append(x)
9     return communs
10 # Exemple
11 s1 = " Python est un langage de programmation de haut
   niveau "
12 s2 = " Python est un langage interprété "
13 print("La liste des mots communs à s1 et s2 est :",
   motsCommuns(s1,s2))

```

Exercice 38. Écrire un programme Python qui permet de chercher le mot le plus long sur une **chaîne s**.

Solution.

```

1 # coding : utf-8
2 def motMax(s) :
3     L = s.split()

```

```

4     mot = ""
5     for x in L :
6         if (len(x)>len(mot)) :
7             mot = x
8     return mot
9 # Exemple
10 s = "Python est un langage de programmation orienté objet"
11 print("Le mot le plus long dans la chaîne s est : ",
        motMax(s))

```

Exercice 39. Écrire un programme Python qui permet de compter le nombres de mots sur un test T. On suppose que le texte est bien formé (un espace après chaque ponctuation et aucun espace avant la ponctuation)

Solution.

```

1 # coding : utf-8
2 T = "Python est un langage de programmation. Python est
   orienté objet"
3
4 # Transformation du texte T en une liste
5 L = T.split()
6
7 # Récupération du nombre d'élément de la liste L
8 nombreMots = len(L)
9 print("Le nombre de mot du texte T est : ", nombreMots)

```

Exercice 40. Écrire un programme Python qui permet d'échanger le premier et le dernier mot. Exemple si s = "Python est un langage de programmation", le programme renvoie la chaîne s2 = "programmation est un langage de Python". On suppose que le texte est bien formé (un espace après chaque ponctuation et aucun espace avant la ponctuation)

Solution.

```

1 # coding : utf-8
2 s = "Python est un langage de programmation"
3
4 # Transformation de la chaîne s en une liste L
5 L = s.split()
6
7 # Récupération du nombre d'élément de la liste L
8 n = len(L)

```

```

9
10 # récupération du premier et dernier élément
11 premier = L[0]
12 dernier = L[n-1]
13
14 # On supprime le premier et le dernier élément de la liste
    L
15 L.pop(n-1)
16 L.pop(0)
17
18 # On reconvertit la liste L en une chaîne
19 s1 = " ".join(L)
20
21 # échanger le premier et le dernier élément dans la chaîne
    s
22 s = dernier + " " + s1 + " " + premier
23 print(s)

```

Exercice 41. Créez une fonction Python, appelée `nombreDivisibles()`, qui s'applique à **une liste** de nombres et **un entier** `n`, et qui renvoie le nombre d'éléments de la liste qui sont divisible par `n`.

Solution.

```

1 # coding : UTF-8
2 def nombreDivisibles(L,n) :
3     i = 0
4     for k in L :
5         if ( k%n == 0 ) :
6             i = i + 1
7     return i
8 L = [12, 4, 7, 9,11]
9 n = 3
10 print("Le nombre d'élément de L qui sont divisible par ",n
    , " est : ",nombreDivisibles(L,n))
11 # ce qui affiche : Le nombre d'élément de L qui sont
    divisible par 3 est : 2

```

Exercice 42. Créer une fonction en Python `nombreOccurences()` qui s'applique à une **liste** `L` et un **élément** `x` comme paramètres et qui retourne le nombre de fois où l'**élément** `x` apparait dans la **liste** `L` sans utiliser la fonction `count()`.

Solution.

```

1 def nombreOccurences(L , x) :
2     occ = 0
3     for element in L :
4         if element == x :
5             occ += 1
6     return occ
7
8 L = [3 , 11 , 3 , 8 , 3 , 23 , 3 , 7 , 11 , 3]
9 print("Le nombre d'occurrences de 3 dans l est : " ,
        nombreOccurences(L,3))

```

Exercice 43. Créer une fonction Python nommée **InsertEtoile()** qui place des 'étoiles' entre les caractères d'une chaîne fournie en entrée.

Exemple pour la chaîne **s** = « Python » , **InsertEtoile(s)** donne **P*t*h*o*n**

Solution.

```

1 # coding : utf-8
2 def insertEtoile(s) :
3     """création d'une chaîne vide qui va contenir
4     les caractères de la chaîne s et séparés par des
5     étoiles
6     """
7     s2 = ""
8     #insérer des * entre les caractères de la chaîne s
9     for x in s :
10         s2 = s2 + x + "*"
11     return s2
12
13 # Exemple :
14 s = "Python"
15 print(insertEtoile(s))

```

Exercice 44. Créer une fonction Python nommée **toutEnMajuscule()** qui permet de transformer une liste formée de chaînes de caractères en une autre liste constituée de chaînes de caractères en **majuscule**. **Exemple** si **L** = ["Python", "est", "un", "langage", "de", "programmation"], la fonction doit retourner la liste **L2** = ["PYTHON", "EST" , "UN", "LANGAGE", "DE", "PROGRAMMATION"].

Solution.


```

1 # coding : utf-8
2 def maj(L) :
3     # Création d'une liste vide qui accueillir les mots en
      majuscule
4     listMaj = []
5
6     # récupération des mots de L et les insérer en
      majuscule dans la liste listMaj
7     for word in L :
8         listMaj.append(word.upper())
9     return listMaj
10
11 # Tester le programme
12 L = ["Python", "est", "un", "langage", "de", "
      programmation"]
13 print(maj(L))

```

Exercice 45. Écrire une fonction en Python qui prends en argument une chaîne de caractères **s** et qui renvoie le nombre de **minuscules** et de **majuscules** contenu dans la chaîne **s**.

Solution.

```

1 # coding : utf-8
2 def nombreMajMin(s) :
3
4     # on initialise le nombre de majuscules et de
      minuscules à zéro
5     nombreMaj = 0
6     nombreMin = 0
7
8
9     # on parcourt les lettres de s en testant si le
      caractère est maj ou min
10    for lettre in s :
11        if(lettre.isupper()) :
12            nombreMaj = nombreMaj + 1
13        elif lettre.islower() :
14            nombreMin = nombreMin + 1
15
16    return (nombreMaj , nombreMin)
17 # On teste l'algorithme
18 s = "Python"
19 print(nombreMajMin(s))
20 print("Le nombre de majuscules est : " , nombreMajMin(s)
      [0])
21 print("Le nombre de minuscules est : " , nombreMajMin(s)
      [1])

```

Exercice 46. Écrire une fonction en langage Python permettant de fournir la liste des chiffres d'un nombre écrit en base 10 sans convertir le nombre en string et sans utiliser aucune fonction prédéfinie.

Solution.

```
1 # Laissé au lecteur !
```

Exercice 47. Écrire un algorithme en langage Python qui permet de renvoyer une liste formée des **mots communs** à deux textes. Exemple si **T1** = "Python est un langage de programmation" et **T2** = "Python est orienté objet", le programme doit renvoyer la liste des mots communs **L** = ['Python', 'est'].

Solution.

```
1 # coding : utf-8
2 def motsCommuns(T1 , T2) :
3
4     # Convertir les textes en des listes
5     L1 = T1.split()
6     L2 = T2.split()
7
8     # initialisation de la liste des mots communs à vide
9     communs = []
10
11    # on parcourt les éléments de la liste L1 et on teste
12    # leurs appartenance à L2
13    for mot in L1 :
14        if mot in L2 :
15            communs.append(mot)
16
17    return communs
18
19 # On teste l'algorithme
20 T1 = "Python est un langage de programmation"
21 T2 = "Python est orienté objet"
22 print(motsCommuns(T1,T2))
23 # La sortie est : ['Python', 'est']
```

Exercice 48. Écrire un programme Python sous forme de fonction qui prend comme paramètre une **chaîne s**, et qui renvoie **'True'** si le premier caractère est identique au dernier caractère de la chaîne, et renvoie **'False'** sinon. Exemple si **s** = "render", la fonction doit renvoyer **True**. Si **s** = "Python", la fonction renvoie **False**.

Solution.

```

1 # coding : utf-8
2 def lastFirst(s) :
3     if len(s) == 0 :
4         return True
5     # on teste si le premier caractère est identique au
    dernier
6     elif s[0] == s[-1] :
7         return True
8     else :
9         return False
10
11 # Exemple
12 print(lastFirst("render")) # affiche : True
13 print(lastFirst("Python")) # affiche : False

```

Exercice 49. Écrire un programme en Python sous forme de fonction qui prend comme paramètre un tuple de chaînes de caractères (s, s_1) et qui retourne l'index de la première occurrence de s_1 trouvée dans la chaîne s sans utiliser la méthode `index()`. La fonction doit renvoyer -1 si l'occurrence s_1 n'est pas trouvée dans la chaîne s . **Exemple** si $s = \text{"Python programming"}$ et $s_1 = \text{"thon"}$, la fonction renvoie l'index 2.

Solution.

```

1 # coding : utf-8
2 def rechercher(s , s1) :
3     n = len(s)
4     m = len(s1)
5     # initialisation de l'index recherché
6     k = -1
7     # parcourir les élément de la chaîne s et rechercher l
    'occurrence s1
8     for i in range(0 , n-m) :
9         if s[i:i+m] == s1 :
10             k = i
11             break
12     return k
13
14 s = "Python Programming"
15 s1 = "thon"
16 print(rechercher(s , s1)) # affiche : 2
17 print(rechercher(s , 'thons')) # affiche : -1

```

Exercice 50. * Écrire un programme Python sous forme de fonction qui prend en paramètre une **chaîne** **s** et qui retourne le premier caractère répété dans la chaîne **s**. Exemple : si **s** = "django framework", la fonction renvoie le caractère 'a'

Solution.

```

1 # coding : utf-8
2
3 # création d'une fonction qui détermine si un caractère
  est répété ou non
4 def isRepeated(s,c) :
5     compteur = 0
6     for x in s :
7         if x == c :
8             compteur = compteur + 1
9         if compteur >= 2 :
10             return True
11     else :
12         return False
13 # fonction qui détermine le premier caractère répété
14 def firstRepeated(s) :
15     repeated = ''
16     for x in s :
17         if isRepeated(s,x) :
18             repeated = x
19             break
20     return repeated
21
22 # Exemple
23 s = "django framework"
24 print("Le premier caractère répété est : " , firstRepeated
    (s))
25 # La sortie est : Le premier caractère répété est : a

```

Exercice 51. Écrire un programme Python sous forme de fonction qui prend en paramètre une **chaîne** **s** et qui renvoie la liste de tous les **caractères répétés** dans la chaîne **s**. Exemple : si **s** = "langage python", la fonction renvoie la liste : ['n', 'a', 'g']

Solution.

```

1 # coding : utf-8
2
3 # création d'une fonction qui détermine si un caractère
  est répété ou non
4 def isRepeated(s,c) :
5     compteur = 0

```

```

6     for x in s :
7         if x == c :
8             compteur = compteur + 1
9         if compteur >= 2 :
10            return True
11        else :
12            return False
13 # fonction qui détermine la liste de tous les caractères
   répétés
14 def listRepeated(s) :
15     # initialisation de la liste des caractères répétés
16     repeated = []
17     for x in s :
18         if isRepeated(s,x) and x not in repeated :
19             repeated.append(x)
20
21     return repeated
22
23 # Exemple
24 s = "python language"
25 print("La liste des caractères répétés est : " ,
   listRepeated(s))
26 # La sortie est :
27 # La liste des caractères répétés est : ['n', 'a', 'g']

```

Exercice 52. Écrire un algorithme en Python qui détermine l'ensemble des caractères qui composent une **chaîne** **s**. **Exemple** si **s = "Python programming"**, l'algorithme renvoie l'ensemble des caractères :

```

1 {'p', 'i', ' ', 'r', 'o', 'm', 't', 'a', 'h', 'y', 'P', 'g',
   ', 'n'}

```

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui renvoie l'ensemble des
   caractères d'une chaîne
3 def characterSet(s) :
4     # initialization de l'ensemble des caractères
5     Set = set({})
6
7     # parcourir les caractères de la chaîne s
8     for x in s :
9         Set.add(x)
10    return Set
11 # Exemple
12 s = "Python programming"
13 print(characterSet(s))

```

```
14 # output : {'p', 'i', ' ', 'r', 'o', 'm', 't', 'a', 'h', 'y', 'P', 'g', 'n'}
```

Exercice 53. Écrire un algorithme en Python qui détermine l'ensemble des mots qui composent une **chaîne texte s**. Exemple si **s = "Python is more power than Java"**, l'algorithme renvoie l'ensemble :

```
1 {'than', 'Python', 'Java', 'power', 'more', 'is'}
```

Solution.

```
1 # coding : utf-8
2 # création d'une fonction qui renvoie l'ensemble des mots
  qui composent une chaîne texte.
3 def wordSet(s) :
4     # initialization de l'ensemble des mots
5     Set = set({})
6
7     # convertir la chaîne en une liste
8     ListWords = s.split()
9     # parcourir les mots de la liste ListWords
10    for word in ListWords :
11        Set.add(word)
12    return Set
13 # Exemple
14 s = "Python is more power than Java"
15 print(wordSet(s))
16 # output : {'is', 'power', 'Java', 'more', 'Python', 'than'}
```

Exercice 54. Écrire un programme Python permettant à partir d'une liste donnée de créer un fichier texte dont les lignes sont les éléments de cette liste. Exemple si la liste est : **List_programming_books = ["Python programming books", "Java programming books", "C ++ programming books", "C # programming books"]**, le fichier généré sera formé par les lignes :

```
Python programming books
Java programming books
C ++ programming books
C # programming books
```

Solution.

```

1 import os
2 List_programming_books = ["Python programming books", "
    Java programming books", "C ++ programming books", "C
    # programming books"]
3 file = open("List_programming_books.txt" , "w")
4 for item in List_programming_books :
5     file.write(item + "\n")
6 file.close()
7 os.startfile("List_programming_books.txt")

```

Exercice 55. Écrire un programme python qui demande à l'utilisateur d'entrer des nombres séparés par des points-virgules ';' et génère une liste composée des nombres saisis.

Solution.

```

1 # coding : utf-8
2 # demander à l'utilisateur de saisir des nombres séparés
    par des point virgule ';'
3 numbers = input("Saisir des nombres séparés par des points
    virgules : ")
4
5 # création de la liste formée des nombre saisis
6 listNumbers = numbers.split(';')
7
8 # afficher le résultat
9 print("Liste des nombres saisis : " , listNumbers)
10 # Exemple pour les nombres saisis : 23;11;47;31;7;18
11 # La sortie est : Liste des nombres saisis : ['23', '11',
    '47', '31', '7', '18']

```

Exercice 56. Écrire un programme en Python sous forme de fonction qui prends en paramètre un tuple formé de deux chaînes (s1 , s2) et qui renvoie la liste des caractères communs à s1 et s2. Exemple : si s1 = 'Python language' et s2 = 'Java Programming', la fonction renvoie la liste : ['P', 'o', 'n', ' ', 'a', 'g']

Solution.

```

1 # coding : utf-8
2 def commonCharacters(s1 , s2) :
3
4     # initialiser la liste des caractères communs à s1 et s2

```

```

5     listCommon = []
6
7     for x in s1 :
8         if x in s2 and x not in listCommon :
9             listCommon.append(x)
10    return listCommon
11
12    s1 = "Python language"
13    s2 = "Java Programming"
14    print(commonCharacters(s1 , s2)) # the output is : ['P', 'o',
    ' ', 'n', ' ', ' ', 'a', 'g']

```

Exercice 57. Écrire un programme en Python sous forme de fonction qui prends en paramètre un tuple formé de deux chaînes (s1, s2) et qui renvoie la liste des caractères de la chaîne s1 qui ne se trouve pas dans la chaîne s2. Exemple : si s1 = 'Python language' et s2 = 'Java Programming', la fonction renvoie la liste : ['y', 't', 'h']

Solution.

```

1  def differenceS1S2(s1 , s2) :
2      listDifference = []
3      for x in s1 :
4          if x not in s2 and x not in listDifference :
5              listDifference.append(x)
6      return listDifference
7
8  s1 = "Python Programming"
9  s2 = "Java Programming"
10 print(differenceS1S2(s1 , s2)) # the output is : ['y', 't',
    'h']

```

Exercice 58. Écrire un programme Python sous forme de fonction qui prend comme paramètres une paire de chaînes (s1, s2) et qui retourne la chaîne s obtenue en concaténant de manière alternative les caractères de s1 et s2. Exemple : pour (s1, s2) = ("Python", "Java"), la fonction renvoie la chaîne s = 'PJyatvha'

Solution.

```

1  # coding : utf-8
2  def concatenation(s1 , s2) :
3      # initialsons la chaîne obtenue par concaténation
    alternative

```



```

4     s = ""
5     n = len(s1)
6     m = len(s2)
7     if n < m :
8         for i in range(0 , n) :
9             s = s + s1[i] + s2[i]
10            s = s + s2[n+1 : m]
11    else :
12        for i in range(0 , m) :
13            s = s + s1[i] + s2[i]
14            s = s + s2[m+1 : n]
15
16    return s
17 s1 = "Python"
18 s2 = "Java"
19 print("la concaténation alternative de s1 et s2 est s = "
        ,concatenation(s1 , s2))

```

Exercice 59. Écrire un programme Python sous forme de fonction qui compte le nombre de fois qu'un caractère apparaît dans une chaîne sans utiliser de fonctions prédéfinies.

Solution.

```

1  # coding : utf-8
2  def caractere_count (s, c) :
3      # initialisation du compteur
4      compteur = 0
5      for x in s :
6          if c == x :
7              compteur = compteur + 1
8      return compteur
9
10 # Exemple de test
11 s = "python programming"
12 print ("Le nombre d'apparitions du caractère 'n' dans 's'
        est =", caractere_count (s, 'n'))
13 # La sortie est : Le nombre d'apparitions du caractère 'n'
        dans 's' est = 2

```

Exercice 60. Écrire un programme en python sous forme de fonction qui prend en paramètre une chaîne s et qui renvoie la liste des caractères numériques contenus dans la chaîne s. Exemple si s = 'Python 3.0, sorti en 2008 et complètement révisé en 2020', la fonction renvoie la liste : [3, 0, 2, 0, 0, 8, 2, 0, 2, 0]

Solution.

```

1 # coding : utf-8
2 def extractNumbers(s) :
3     # initialiation de la liste des nombres à extraire
4     L = []
5
6     for x in s :
7         if x.isnumeric() :
8             x = int(x)
9             L.append(x)
10    return L
11 s = 'Python 3.0, sorti en 2008 et entièrement révisé en
    2020'
12 print(extractNumbers(s))
13 # La sortie est : [3, 0, 2, 0, 0, 8, 2, 0, 2, 0]

```

Exercice 61. Écrire un programme Python qui détermine la liste de tous les caractères d'occurrence maximum dans une chaîne donnée. Exemple : si `s = "Programming"`, l'algorithme renvoie la liste : `['r', 'g', 'm']`

Solution.

```

1 # Laissé au lecteur

```

Exercice 62. En utilisant le **code ascii**, écrire un programme python qui renvoie et affiche la liste de tous les caractères majuscules `[A, B, C, ..., Z]` et la liste de tous les caractères minuscules `[a, b, c, .., z]`

Solution.

- Les codes ascii pour les caractères A, B, C, ..., Z sont : 65, 66, ..., 90
- Les codes ascii pour les caractères a, b, c, ..., z sont : 97, 98 ..., 122
- Pour obtenir un caractère du code ascii, nous utilisons la méthode `chr()`

Et par suite :

```

1 #coding : utf-8
2 list_uppercase_characters = [chr(i) for i in range(65 ,
    91)]
3 list_lowercae_characters = [chr(i) for i in range(97 ,
    122)]
4 print("La liste des caractères en majuscule est : " ,
    list_uppercase_characters)

```

```

5 print("La liste des caractères en minuscule est : " ,
      list_lowercase_characters)
6 #La sortie est :
7 """
8 La liste des caractères en majuscule est :
9 ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
10  'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
11  'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
12 La liste des caractères en minuscule est :
13 ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
14  'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
15  's', 't', 'u', 'v', 'w', 'x', 'y']
16 """

```

Exercice 63. En utilisant l'exercice précédent (Exercice 62), créez un programme python sous forme de fonction qui prend en paramètre une chaîne `s` et qui renvoie la même chaîne en majuscules sans utiliser la méthode `upper()`. Vous pouvez utiliser les méthodes `chr()` et `ord()`.

Solution.

```

1 #coding : utf-8
2 def toUppercase(s) :
3     # En utilisant l'exercice 62, on obtient facilement la
4     # listes des caractères en majuscule
5     #et la liste des caractères en minuscule :
6     uppercase_characters = [chr(i) for i in range(65 , 91)
7                             ]
8     lowercase_characters = [chr(i) for i in range(97 ,
9                             122)]
10    # on peut donc obtenir un caractère majuscule à partir
11    # d'un caractère minuscule
12    #en soustrayant 32 à son code ascii
13
14    # initialisation de la liste qu'on cherche
15    s_upper = ""
16    for x in s :
17        if x in lowercase_characters :
18            # on transforme le caractère minuscule en
19            # majuscule
20            x = chr(ord(x) - 32)
21            # on ajoute le caractère à la liste
22            s_upper = s_upper + x
23        else :
24            s_upper = s_upper + x
25
26    return s_upper
27 print(toUppercase("Python Programming"))
28 # La sortie est : PYTHON PROGRAMMING

```

Exercice 64. Écrire un programme python comme une fonction qui prend comme paramètre une chaîne s et qui retourne une autre chaîne obtenue à partir de s en supprimant les espaces au début de la chaîne s et les espaces à la fin de la chaîne s sans utiliser la méthode `lstrip()` ni aucune méthode prédéfinie. Exemple si s = " Bonjour " , la fonction renvoie la chaîne "Bonjour"

Solution.

```

1
2 # coding : utf-8
3 def removeSpace(s) :
4     n = len(s)
5     # initialisation du nombre d'espace à la fin de la
    chaîne s
6     j = 0
7     while (s[n-1-j]) == " " :
8         j = j + 1
9     s = s[:n-j]
10    # initialisation du nombre d'espace au début de la
    chaîne s
11    i = 0
12    while s[i] == " " :
13        i = i + 1
14    s = s[i:]
15    return s
16
17 # Exemple
18 s = "          Hello World          "
19 print("'%',s,'%'") # affiche : '          Hello World
20 print("'%',removeSpace(s),'%'") # affiche : 'Hello World'

```

Exercice 65. Écrire un programme python sous forme de fonction qui prend en paramètre une chaîne s et qui retourne une chaîne obtenue à partir de la chaîne s en transformant chaque caractère majuscule en caractère minuscule et vice versa sans utiliser la méthode `swapcase()`. Exemple si s = "Hello Wordl!" , la fonction renvoie la chaîne "hELLO wORLD!"

Solution.

```

1 #coding : utf-8
2 def mySwapcase(s) :
3
4     # initialize the swapped string
5     s_swap = ""
6

```

```

7     # parcourir les éléments de s
8     #and testing if the character is in uppercase or
      lowercase
9     for x in s :
10        if x.isupper() :
11
12            # transformer les caractères majuscules en
            minuscules
13            x = x.lower()
14            s_swap = s_swap + x
15
16        elif x.islower() :
17            # transformer les caractères minuscules en
            majuscules
18            x = x.upper()
19            s_swap = s_swap + x
20        else :
21            s_swap = s_swap + x
22
23    return s_swap
24
25 # Exemple
26 print(mySwapcase("Hello World !")) # La sortie est : hELLO
      wORLD!

```

Exercice 66. Écrire un algorithme python sous forme de fonction qui prend en paramètre une **chaîne de caractères s** et qui renvoie la **liste** des **mots** contenant au moins un caractère non alphabétique. Exemple pour **s = "C# is syntactically similar to Java2"** , la fonction renvoie la liste : ["C#", "Java2"]

Solution.

```

1  # fonction qui renvoie le nombre de caractères non
      alphabetiques dans une chaîne
2  def not_alphabetical(s) :
3      # initialiser la liste des mots demandée
4      L_word_not_alphabet = []
5
6      # convertir s en une liste
7      list_word = s.split()
8
9      for word in list_word :
10         if word.isalpha() == False :
11             L_word_not_alphabet.append(word)
12     return L_word_not_alphabet
13 #Exemple
14 s = "C# is syntactically similar to Java2"

```

```
15 print(not_alphabetical(s)) # affiche : ['C#', 'Java2']
```

Exercice 67. Écrire un programme python sous forme de fonction qui prend en paramètre un tuple de **chaîne** (**s**, **s1**) et qui retourne l'index de la première occurrence de **s1** trouvée dans la chaîne **s** sans utiliser aucune fonction prédéfinie. La fonction doit retourner -1 si **s1** n'est pas trouvé dans la chaîne **s** Exemple si **s** = "Python language" et **s1** = "lan", la fonction renvoie l'index 7.

Solution.

```
1 #coding : utf-8
2 def Find(s , s1) :
3     n = len(s)
4     m = len(s1)
5     k = -1
6     for i in range(0 , n) :
7         if s[i:i+m] == s1 :
8             k = i
9             break
10    return k
11
12 s = "Python language"
13 s1 = "lang"
14 print(Find(s , s1))      # affiche 7
15 print(Find(s , 'land')) # affiche -1
```

Exercice 68. Écrire un algorithme python qui calcul le nombre de caractères communs à deux chaînes **s1** et **s2**. Exemple si **s** = "Hello" et **s2** = "World", les caractères communs sont 'l' et 'o', et par suite l'algorithme renvoie 2.

Solution.

```
1 s1 = "Hello"
2 s2 = "World"
3
4 # initialisation du compteur
5 compteur = 0
6
7 # création et initialisation d'une chaîne en supprimant
  les caractères répétés dans s1
```

```

8 s = ""
9 for x in s1 :
10     if x not in s :
11         s = s + x
12 for x in s :
13     if x in s2 :
14         compteur = compteur + 1
15
16 # afficher le nombre de caractères communs à s1 and s2
17 print(compteur) # affiche 2

```

Exercice 69. Écrire un algorithme python qui renvoie pour une chaîne de caractères donnée **s** le tuple (**maj** , **min**) avec **maj** est le nombre de majuscules dans **s** et **min** le nombre de minuscules dans **s**.

Solution.

```

1 # coding : utf-8
2
3 def majMin(s) :
4     # initialiser le nombre de majuscules et le nombre de
5     # minuscules
6     nombre_maj , nombre_min = 0 , 0
7     for x in s :
8         if x.isupper() :
9             nombre_maj = nombre_maj + 1
10        if x.islower() :
11            nombre_min = nombre_min + 1
12    return (nombre_maj , nombre_min)
13
14 # Exemple
15 s = "Django Framework"
16 print(majMin(s)) # affiche (2, 13)

```

Exercice 70. Écrire un programme en python qui extrait d'une chaîne donnée **s**, la liste de tous les mots dont la longueur est inférieure ou égale à 4. Exemple si **s** = "Le langage de programmation Python est open source et très facile à apprendre", le programme doit retourner la liste : **L** = ['Le', 'de', 'est', 'open', 'et', 'très', 'à']

Solution.

```

1 # coding : utf-8
2 def listWords(s) :

```

```

3
4     # initialisation de la liste qu'on souhaite obtenir
5     list_word = []
6     # convertir la chaîne s en une liste
7     L = s.split()
8     for word in L:
9         if (len(word) <= 5):
10             list_word.append(word)
11     return list_word
12
13 # Exemple
14 s = "Le langage de programmation Python est open source et
    très facile à apprendre"
15 print(listWords(s)) # affiche : ['Le', 'de', 'est', 'open',
    'et', 'très', 'à']

```

Exercice 71. Écrire un programme Python qui détermine le chiffre **minimum** dans une chaîne fr caractères donnée s. Exemple si s = "**Python3.7 est plus puissant que Python2.7**", l'algorithme doit renvoyer le chiffre 2. Nous supposons que la chaîne s ne contient aucun nombre négatif.

Solution.

```

1 def minimumDigit(s):
2     # création & initialisation de la liste des chiffres
3     listDigits = []
4     # parcourir les caractères de s et extraire les
5     chiffres
6     for x in s:
7         if x.isdigit():
8             x = int(x)
9             listDigits.append(x)
10
11     # initialisation du minimum des chiffres
12     minDigit = listDigits[0]
13
14     # parcourir les chiffres de la liste et extraire le
15     minimum
16     for n in listDigits:
17         if minDigit > n:
18             minDigit = n
19     return minDigit
20 # Exemple :
21 s = "Python3.7 is more power than Python2.7"
22 print("Le minimum ds chiffre dans s est :", minimumDigit(
    s))
23 # La sortie est : Le minimum des chiffre dans s est : 2

```


Exercice 72. Écrire un algorithme en Python qui permet d'examiner si une occurrence est présente dans une chaîne de caractères donnée ou non.

Solution.

```

1 # coding : utf-8
2 def examineOccurrence(s , occ) :
3     # obtenir la longueur de l'occurrence occ et la
4     # longueur de la chaîne s
5     m = len(occ)
6     n = len(s)
7
8     # initialize counter
9     counter = 0
10
11    # recherche de l'occurrence dans la chaîne s
12    for i in range(0 , n-m) :
13        if s[i : m + i] == occ :
14            counter = counter + 1
15
16    if counter > 0 :
17        return True
18    else :
19        return False
20
21    # Exemple :
22    s = "Python is the most popular programming language"
23    occ1 = "most"
24    occ2 = "algorithm"
25    print(examineOccurrence(s, occ1)) # affiche : True
26    print(examineOccurrence(s, occ2)) # affiche : False

```

Exercice 73. Écrire un algorithme python qui **supprime** toutes les **voyelles** d'une chaîne de caractères s. Exemple si s = "Python is hight level programming language", l'algorithme renvoie la chaîne : "Pthn s hght lvl prgrmmng lngg"

Solution.

```

1 # coding : utf-8
2 def deleteVowels(s) :
3     # création de la liste des voyelles
4     vowels = ['a','e','y','u','i','o']
5
6     # initialisation de la chaîne sans vowels
7     s1 = ""
8
9     # parcourir les caractères de la chaîne s et
10    # élimination des voyelles

```

```

10     for x in s :
11         if x not in vowels :
12             s1 = s1 + x
13     return s1
14
15 # Exemple
16 s = "Python is hight level programming language"
17 print(deleteVowels(s))
18 # affiche : Pthn s hght lvl prgrmmng lngg

```

Exercice 74. Écrire un programme Python qui remplace les caractères d'index impaires d'une chaîne donnée par '#'. Exemple : si `s = "Python"`, l'algorithme renvoie la chaîne : `"P#t#o#"`

Solution.

```

1 # coding : utf-8
2 def replace(s) :
3     n = len(s)
4
5     # initialisation de la chaîne recherchée
6     new_string = ""
7
8     # parcourir les caractères de la chaîne s
9     for i in range(0 , n) :
10
11         # ajouter uniquement les caractères d'index pairs
12         if( i%2 == 0 ) :
13             new_string = new_string + s[i]
14         else :
15             new_string = new_string + '#'
16     return new_string
17
18 # Exemple
19 s = "Python"
20 print(replace(s)) # affiche : P#t#o#

```

Exercice 75. Écrire un algorithme python qui détermine la liste de tous les **caractères communs** à deux chaînes de caractères `s1` et `s2` sans répétition. Exemple : si `s1 = "langage Python"` et `s2 = "Programmation"`, l'algorithme renvoie la liste : `['P', 'o', 'n', 'a', 'g']` (le caractère 'g' doit être ajouté une seule fois même si partagé deux fois)

Solution.

```

1 # coding :utf-8
2 def commonCharacters(s1 , s2) :
3     # initialisation de la liste des caractères communs
4     listCommon = []
5     for x in s1 :
6         if x in s2 and x not in listCommon :
7             listCommon.append(x)
8     return listCommon
9
10 # Example
11 s1 = "Python language"
12 s2 = "Programming"
13 print("La liste de caractère communs est : ",
14       commonCharacters(s1 , s2))
14 # La liste de caractère communs est :  ['P', 'o', 'n', 'a',
15                                         ', 'g']

```

Exercice 76. Écrire un algorithme python qui transforme une chaîne de caractères donnée **s** en échangeant le deuxième caractère (**s[1]**) avec l'avant dernier caractère. Nous supposons que **len(s)** est supérieur ou égal à 4. Exemple : si **s = "Python"**, l'algorithme renvoie la chaîne : **"Pothyn"**.

Solution.

```

1 def swapping(s) :
2
3     n = len(s)
4     s_swap = s[0] + s[n-2] + s[2:n-2] + s[1] + s[n-1]
5
6     return s_swap
7
8 # Exemple
9 s = "Python"
10 print(swapping(s)) # affiche : Pothyn

```

Exercice 77. Écrire un algorithme Python qui détermine le premier index d'une occurrence existante dans une chaîne de caractères donnée **s** sans utiliser les méthodes prédéfinies comme **find()** ou **rfind()** ... L'algorithme doit renvoyer -1 si l'occurrence n'existe pas dans la chaîne **s**. Exemple : si **s = "langage de programmation Python"** et **occ = "prog"** l'algorithme renvoie **7**

Solution.

```

1 # coding:utf-8
2 def findFirstOccurrence(s , occ):
3     # obtenir les longueurs des chaînes occ et s
4     m = len(occ)
5     n = len(s)
6
7     # initialisation de lindex
8     index = -1
9     # recherche de l'occurrence dans la chaîne s
10    for i in range(0 , n-m):
11        if s[i : m + i] == occ:
12            index = i
13            break
14    return index
15
16 # Exemple :
17 s = "Python programming language"
18 occ1 = "prog"
19 occ2 = "algorithm"
20 print(findFirstOccurrence(s, occ1)) # affiche : 7
21 print(findFirstOccurrence(s, occ2)) # affiche : -1

```

Exercice 78. Écrire un algorithme python qui retourne la liste de tous les index où une occurrence **occ** est trouvée dans une chaîne donnée **s** sans utiliser aucune méthodes prédéfinies comme **find()**, **rfind()**, **index()** ... La fonction renvoie la liste vide [], si l'occurrence **occ** n'existe pas dans **s**. Exemple : si **s = Python is an interpreted language. Python is open source. Python is easy to learn** **occ = "Python"**, la fonction renvoie la liste : [0, 35, 58]

Solution.

```

1 def indexOfOccurrence(s , occ):
2     # obtenir la longueur de l'occurrence occ et la
3     # longueur de la chaîne s
4     m = len(occ)
5     n = len(s)
6
7     # initialisation de la liste des index
8     listIndex = []
9     # rechercher les index de toutes le occurrences occ
10    dans s
11    for i in range(0 , n-m):
12        if s[i : m + i] == occ:
13            listIndex.append(i)
14    return listIndex

```

```

14 # Exemple :
15 s = "Python is an interpreted language. Python is open
    source. Python is easy to learn"
16 occ = "Python"
17 print(indexOfOccurrence(s, occ))
18 # La sortie est : [0, 35, 58]

```

Exercice 79. ...

Solution.

Exercice 80. Écrire un algorithme python qui détermine la liste de tous les mots communs à deux textes T1 et T2 sans répétition. Exemple si : T1 = "Python is open source programming language. Python was created on 1991" et T2 = "Python is the most popular programming language ", l'algorithme renvoie la liste : ['Python', 'is', 'programming', 'language.']

Solution.

```

1 def commonWords(T1 , T2) :
2     # convertir les textes en des listes
3     listWords_in_T1 = T1.split()
4     listWords_in_T2 = T2.split()
5
6     # initialisation de la liste des mots communs
7     listCommon_words = []
8
9     # rechercher les mots communs à T1 et T2 et les
10    ajouter à la liste listCommon_words
11    for word in listWords_in_T1 :
12        if word in listWords_in_T2 and word not in
13        listCommon_words :
14            listCommon_words.append(word)
15    return listCommon_words
16
17 # Exemple
18 T1 = "Python is open source programming language. Python
19    was created on 1991"
20 T2 = "Python is the most popular programming language. "
21 print("La liste des mots communs est : " , commonWords(T1
22    , T2))
23
24 # La sortie est :
25 # La liste des mots communs est : ['Python', 'is', '
26    programming', 'language.']

```

Exercice 81. Écrire un algorithme Python sous forme de fonction qui prend en paramètre une **chaîne de caractères** **s** et qui renvoie le **dictionnaire** dont les clés sont les mots qui composent la chaîne **s** et dont les valeurs des clés sont les nombres d'occurrences des mots dans la chaîne **s**. Exemple : si **s = "I use Python for datascience but I don't use Python for mobile"**, l'algorithme renvoie le dictionnaire :

```
1 d = { 'I': 2, 'use': 2, 'Python': 2, 'for': 2, 'datascience': 1, 'but': 1, "don't": 1, 'mobile': 1 }
```

Solution.

```
1 # coding : utf-8
2 def wordOccurrence(s) :
3
4     # initialisation du dictionnaire qu'on souhaite
    obtenie
5     d = dict({})
6
7     # convertir la chaîne s en une liste
8     listWords = s.split()
9
10    # parcourir les éléments de la liste listWords
11    # ajouter ensuite le nombre d'occurrences de chaque
    mot
12    for word in listWords :
13        d[word] = listWords.count(word)
14
15    return d
16
17 # Exemple
18 s = "I use Python for datascience but I don't use Python
    for mobile"
19 print(wordOccurrence(s))
20 # La sortie est :
21 #{'I': 2, 'use': 2, 'Python': 2, 'for': 2, 'datascience':
    1, 'but': 1, "don't": 1, 'mobile': 1}
```

Exercice 82. Écrire un algorithme Python qui supprime toutes les chaînes vides d'une liste de chaînes. Exemple : Si : **L = ["Python" , "" , "is" , "" , "the", "most" , "" , "used" , "programming", "language" , ""]**, l'algorithme renvoie la liste : **['Python', 'is', 'the', 'most', 'used', 'programming', 'language']**

Solution.

```

1 # coding : utf-8
2 def remove_empty_string(L) :
3     # initialisation de la liste sans chaîne vide
4     newList = []
5
6     # parcourir les éléments de la liste L
7     for word in L :
8         if word != "" :
9             newList.append(word)
10    return newList
11
12 # Exemple
13 L = [ "Python" , "" , "is" , "" , "the" , "most" , "" , "
14     used" , "programming" , "language" , "" ]
15 print(remove_empty_str(L))
16 # La sortie est :
17 # ['Python', 'is', 'the', 'most', 'used', 'programming', '
18     language']

```

Exercice 83. Écrire un programme Python sous forme de fonction qui prend comme paramètre une chaîne `s` et qui renvoie le premier caractère répété dans la chaîne `s`. Exemple : si `s = "django framework"`, la fonction renvoie le caractère `'a'`

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui teste si un caractère donné
3   est répété dans une chaîne donnée
4 def isRepeated(s,c) :
5     counter = 0
6     for x in s :
7         if x == c :
8             counter = counter + 1
9     if counter >= 2 :
10        return True
11    else :
12        return False
13 # fonction qui détermine le premier caractère répété
14 def firstRepeated(s) :
15     repeated = ''
16     for x in s :
17         if isRepeated(s,x) :
18             repeated = x
19             break
20     return repeated
21 # Exemple

```

```
22 s = "django framework"
```

Exercice 84. Écrire un programme Python sous forme de fonction qui prend comme paramètre une chaîne de caractères `s` et qui renvoie la liste de tous les caractères répétés dans la chaîne `s` sans utiliser aucune méthode ni aucun module prédéfini en Python. Exemple : si `s = "Python langage"`, la fonction renvoie la liste : `['n', 'a', 'g']`

Solution.

```
1
2 # coding : utf-8
3 # création d'une fonction qui teste si un caractère donné
  est répété dans une chaîne donnée
4 def isRepeated(s,c) :
5     counter = 0
6     for x in s :
7         if x == c :
8             counter = counter + 1
9     if counter >= 2 :
10        return True
11    else :
12        return False
13
14
15 # fonction qui détermine le premier caractère répété
16 def listRepeated(s) :
17     repeated = []
18     for x in s :
19         if isRepeated(s,x) and x not in repeated :
20             repeated.append(x)
21
22     return repeated
23
24 # Exemple
25 s = "python language"
26 print("La liste des caractères répétés est : " ,
      listRepeated(s))
27 # La sortie est :
28 # La liste des caractères répétés est :  ['n', 'a', 'g']
```

Exercice 85. Écrire un algorithme python qui détermine l'ensemble de tous les caractères qui composent une chaîne `s`. Exemple si `s = "programmation Python"`, l'algorithme renvoie l'ensemble :


```
1 {'p', 'i', ' ', 'r', 'o', 'm', 't', 'a', 'h', 'y', 'P', 'g',
  ' ', 'n'}
```

Solution.

```
1
2 # coding : utf-8
3 def characterSet (s) :
4     # initialisation de l'ensemble des caractères qui
      compose la chaîne
5     Set = set ({})
6
7     # parcourir les caractères de s
8     for x in s :
9         Set.add (x)
10    return Set
11
12 # Exemple
13 s = "Python programming"
14 print (characterSet (s))
15 # La sortie est : {'g', 't', 'y', 'a', 'P', 'm', ' ', 'n',
      'h', 'p', 'o', 'r', 'i'}
```

Exercice 86. Écrire un algorithme en Python qui détermine l'ensemble des mots qui composent un texte T. Exemple si **T = "Python is more power than Java"**, l'algorithme renvoie l'ensemble :

```
1 {'than', 'is', 'Java', 'more', 'Python', 'power'}
```

Solution.

```
1 # coding : utf-8
2 # fonction qui renvoie l'ensemble des mots qui composent
      une chaîne
3 def wordSet (T) :
4
5     # initialisation de l'ensemble des mots
6     Set = set ({})
7
8     # convertir la chaîne en une liste
9     ListWords = T.split ()
10
11    # parcourir les mots de la liste ListWords
12    for word in ListWords :
13        Set.add (word)
14
```

```

15     return Set
16
17 # Exemple
18 T = "Python is more power than Java"
19 print (wordSet (T))
20 # la sortie est : {'power', 'is', 'than', 'Python', 'Java',
    'more'}

```

Exercice 87. Écrire un algorithme Python sous forme de fonction qui prend en paramètre un tuple de deux chaînes (**s1**, **s2**) et qui renvoie la liste des caractères communs à s1 et s2. Exemple : si **s1** = **'Python language'** et **s2** = **'Java Programming'**, la fonction renvoie la liste : **['P', 'o', 'n', ' ', 'a', 'g']**

Solution.

```

1 def commonCharacters(s1 , s2) :
2     listCommon = []
3     for x in s1 :
4         if x in s2 and x not in listCommon :
5             listCommon.append(x)
6     return listCommon
7
8 s1 = "Python language"
9 s2 = "Java Programming"
10 print (commonCharacters(s1 , s2))
11 # la sortie est : ['P', 'o', 'n', ' ', 'a', 'g']

```

Exercice 88. Écrire un programme Python sous forme de fonction qui prend en paramètre une chaîne de caractère s et qui renvoie le premier mot répété dans la chaîne s. Exemple : si **s** = **"python programming language, is the most popular programming language"**, la fonction renvoie le mot : **'programming'**.

Solution.

```

1 # Laissé au lecteur

```

Exercice 89. Écrire un algorithme Python qui détermine la liste des caractères répétés dans une chaîne s donnée. Exemple : si **s** = **"Programming language"**, l'algorithme renvoie la liste : **['r', 'g', 'a', 'm', 'n']**

Solution.

```

1 # coding : utf-8
2 # fonction qui permet de tester si un caractères est
  répété ou non
3 def isRepeated(s,c) :
4     counter = 0
5     for x in s :
6         if x == c :
7             counter = counter + 1
8     if counter >= 2 :
9         return True
10    else :
11        return False
12 # création d'une fonction qui renvoie la liste de tous les
  caractères répétés
13 def repeated(s) :
14     # initialisation de la liste des caractères répétés
15     listRepeated = []
16     for x in s :
17         if isRepeated(s , x) and x and x != " " and x not
18         in listRepeated :
19             listRepeated.append(x)
20     return listRepeated
21 # Exemple
22 s = "programming language"
23 print("la liste des caractères répétés dans s est : " ,
  repeated(s))
24 # La sortie est : la liste des caractères répétés dans s
  est : ['r', 'g', 'a', 'm', 'n']

```

Exercice 90. Écrire un programme Python qui détermine la liste de tous les caractères d'occurrence maximum dans une chaîne donnée s. Exemple : si s = "Programming", l'algorithme renvoie la liste : ['r', 'g', 'm']

Solution.

```

1 # Exercice laissé au lecteur !

```

Exercice 91. (**) Écrire un algorithme python qui détermine la liste des mots de longueurs maximales communs à deux chaînes de caractères T1 et T2. Exemple si T1 = 'Python created by Guidorossu is open source programming language. Python was created on 1991'

et `T2 = "Python created by Guidorossu is the most popular programming language Guidorossu"`, l'algorithme renvoie la liste : `['Guidorossu', 'programming']`

Solution.

```

1
2 # coding : utf-8
3 #création d'une méthode qui détermine la liste des mots
  communs en T1 et T2
4 def commonWords(T1 , T2) :
5     # # convertir les textes en listes
6     listWords_in_T1 = T1.split()
7     listWords_in_T2 = T2.split()
8
9     # initialisation de la liste des mots communs
10    listCommon_words = []
11    for word in listWords_in_T1 :
12        if word in listWords_in_T2 and word not in
listCommon_words :
13            listCommon_words.append(word)
14    return listCommon_words
15
16 # Création d'une méthode qui détermine le mot commun avec
  une longueur maximale
17 def commonMax(T1, T2) :
18    listCommon_words = commonWords(T1 , T2)
19    # intitialisation de la longueur maximale
20    maxLength = 0
21    for word in listCommon_words :
22        if len(word) >= maxLength :
23            maxLength = len(word)
24
25    # recherche des mots communs de longueurs maximales
26    listCommonWordMax = []
27    for word in listCommon_words :
28        if len(word) == maxLength :
29            listCommonWordMax.append(word)
30    return listCommonWordMax
31
32 T1 = "Python created by Guidorossu is open source
  programming language. Python was created on 1991"
33 T2 = "Python created by Guidorossu is the most popular
  programming language Guidorossu"
34 print(commonMax(T1 , T2))
35 # La sortie est : ['Guidorossu', 'programming']

```

Exercice 92. Écrire un programme python sous forme de fonction qui

prend en paramètre une chaîne `s` et retourne `True` si la chaîne `s` contient au moins un caractère majuscule et `False` dans le cas contraire.

Solution.

```

1
2 # coding : utf-8
3 def containsUpper(s) :
4
5     # définir et initialiser une variable qui compte le
      nombre de majuscules
6     counter = 0
7
8     # parcourir les caractères de la chaîne s
9     for x in s :
10         # incrémenter le compteur à chaque fois que le
            caractère rencontré est en majuscule
11         if x.isupper() :
12             counter = counter + 1
13     if counter > 0 :
14         return True
15     else :
16         return False
17 # Exemple
18 print(containsUpper("djangoFramework")) # affiche True
19 print(containsUpper("djangoframework")) # affiche false

```

Exercice 93. Écrire une fonction nommée `compte_les_caracteres()` qui prends en paramètres une chaîne de caractères et qui renvoie l'occurrence des caractères contenus dans la chaîne sous forme de dictionnaire. Exemple : pour `s = "Programmation en Python"` la fonction renvoie le dictionnaire :

```

1 { 'P':2, 'r':2, 'o':3, 'g':1, 'a':2, 'm':2, 't':2, 'i':1, '
      n':3, 'e':1, 'y':1, 'h':1}

```

Solution.

```

1 # coding : utf-8
2 def compte_les_caracteres(s) :
3     d = dict({})
4     listUnique = []
5     for c in s :
6         if c not in listUnique and c != ' ':
7             listUnique.append(c)
8             d[c] = s.count(c)
9

```

```

10     return d
11 # Exemple
12 s = "Programmation en Python"
13 print(compte_les_caracteres(s))
14 # La sortie est :
15 #{'P': 2, 'r': 2, 'o': 3, 'g': 1, 'a': 2, 'm': 2, 't': 2,
    'i': 1, 'n': 3, 'e': 1, 'y': 1, 'h': 1}

```

Exercice. (93 bis) Écrire une fonction nommée `compte_mots_ligne()` qui prends en paramètres une chaîne de caractères et qui renvoie l'occurrence des mots contenus dans la chaîne sous forme de dictionnaire. Exemple pour `s = "Python est langage de programmation. Python est le langage le plus utilisé"`, la fonction renvoie le dictionnaire :

```

1 {'Python': 2, 'est': 2, 'langage': 2, 'de': 1, 'programmation.' :
  '': 1, 'le': 2, 'plus': 1, 'utilisé': 1}

```

Solution.

```

1 # coding: utf-8
2 def compte_mots_ligne(s):
3     d = dict({})
4     listUnique = []
5     listMots = s.split()
6     for mot in listMots:
7         if mot not in listUnique:
8             listUnique.append(mot)
9             d[mot] = listMots.count(mot)
10
11     return d
12 # Exemple
13 s = "Python est langage de programmation. Python est le
    langage le plus utilisé"
14 print(compte_mots_ligne(s))
15 # La sortie est :
16 {'Python': 2, 'est': 2, 'langage': 2, 'de': 1, 'programmation.' :
    '': 1, 'le': 2, 'plus': 1, 'utilisé': 1}

```

Exercice 94. Écrire un algorithme en Python sous forme de fonction qui prends en paramètre une variable chaîne de caractères et qui renvoie la liste des mots qui commencent par une majuscule. Exemple si `s = 'Python is more power than Java and C++'`, la fonction renvoie la liste : `L = ['Python', 'Java', 'C++']`

Solution.

```

1 #coding : utf-8
2 def listMotMaj(s) :
3     # initialisation de la liste des mots qui commencent
      par une majuscule
4     listMaj = []
5     # Convertir la chaîne s en une liste
6     L = s.split()
7     # parcourir les mots de la liste L
8     for mot in L :
9         if (mot[0].isupper()) :
10             listMaj.append(mot)
11     return listMaj
12
13 # Exemple
14 s = 'Python is more power thant Java and C++'
15 print("La liste des mots qui commencent par une majuscule
      est : " , listMotMaj(s))
16 # La sortie est : La liste des mots qui commencent par une
      majuscule est : ['Python', 'Java', 'C++']

```

Exercice 95. Écrire un algorithme en Python sous forme de fonction qui prends en paramètre une chaîne de caractères et qui renvoie la liste des mots qui ne contiennent aucune majuscule. Exemple si `s = 'Python est plus populaire que Java'` , la fonction renvoie la liste : `L = ['est' , 'plus' , 'populaire' , 'que']`

Solution.

```

1 #coding : utf-8
2 def list_mot_sans_maj(s) :
3     # initialisation de la liste des mots sans aucune
      majuscule
4     listSansMaj = []
5
6     # convertir la chaîne s en une liste
7     L = s.split()
8
9     # parcourir les mots de la liste L
10    for mot in L :
11        # tester si l'élément 'mot' contient ou non une
      majuscule
12        if mot.islower() :
13            listSansMaj.append(mot)
14    return listSansMaj
15
16 # Exemple

```

```

17 s = 'Python est plus populaire que Java '
18 print("La liste des mots sans majuscule est : " ,
      list_mot_sans_maj(s))
19 # La sortie est :
20 # La liste des mots sans majuscule est : ['est', 'plus',
      'populaire', 'que']
21 # La sortie est :
22 # La liste des mots sans majuscule est : ['est', 'plus',
      'populaire', 'que']

```

Exercice 96. Écrire un algorithme en Python sous forme de fonction qui prends en paramètre une chaîne de caractères et qui renvoie la liste des mots qui ne contiennent au moins deux majuscules. Exemple si `s = 'La bibliothèque GUI PySide est plus populaire que PyQt'`, la fonction renvoie la liste : `['GUI', 'PySide', 'PyQt']`

Solution.

```

1 # création d'une fonction qui renvoie le nombre de
  majuscule dans une chaîne
2 def nombreMaj(s) :
3     # définir et initialiser une variable qui compte le
      nombre de majuscules
4     counter = 0
5
6     # parcourir les caractères de la chaîne s
7     for x in s :
8         # incrémenter le compteur à chaque fois que le
          caractère rencontré est en majuscule
9         if x.isupper() :
10             counter = counter + 1
11     return counter
12
13 # Fonction qui renvoie la liste des mots qui contiennent
    au moins 2 majuscules
14 def motsDeuxMajuscules(s) :
15     # initialisation de la liste des mots sans aucune
      majuscule
16     listDeuxMaj = []
17
18     # convertir la chaîne s en une liste
19     L = s.split()
20
21     # parcourir les mots de la liste L
22     for mot in L :
23         # tester si l'élément 'mot' contient ou non une
          majuscule
24         if nombreMaj(mot) >= 2 :

```



```

25         listDeuxMaj.append(mot)
26     return listDeuxMaj
27
28 # Exemple
29 s = 'La bibliothèque GUI PySide est plus populaire que
    PyQt'
30 print("La liste des mots ayant au moins 2 majuscules est :
    ", motsDeuxMajuscules(s))
31 # La sortie est :
32 # La liste des mots ayant au moins 2 majuscules est : ['
    GUI', 'PySide', 'PyQt']

```

Exercice 97. Écrire un algorithme en Python sous forme de fonction qui prends en paramètre un **couple (s,x)** formé d'une chaîne de caractères s et un caractère x et qui renvoie l'index de la deuxième position du caractère x dans la chaîne s sans utiliser aucune fonction prédéfinie. La fonction doit renvoyer -1 si le caractère x ne figure pas dans la chaîne s. Exemple si s = 'langage python' et x = 'a', la fonction renvoie l'index : 4

Solution.

```

1  #coding : utf-8
2  def listPositions(s , x):
3      # initialiser la liste des positions du caractère x
    dans la chaîne s
4      listPos = []
5
6      # parcourir les caractères de la chaîne s
7      for i in range(0 , len(s)) :
8          if s[i] == x:
9              listPos.append(i)
10         if len(listPos) >= 2 :
11             return listPos[1]
12         else :
13             return -1
14
15 # Exemple
16 s , x = 'langage python' , 'a'
17 print("La position du caractère " , x , " dans s est : " ,
    listPositions(s , x))
18 # La sortie est :
19 # La position du caractère a dans s est : 4

```

Exercice 98. Écrire un algorithme en Python sous forme de fonction qui permet de transformer les chaînes de caractères en **majuscule sans utiliser la méthode upper()**.

Solution. Rappel sur les fonctions `ord()` et `chr()`

- la fonction `ord()` renvoie le **code ASCII** d'un caractère. Pour les caractère minuscule si on applique cette fonction on obtient :

`ord('a') = 97` , `ord('b') = 98` , ... , `ord('z') = 122`

Tandis que pour les caractères majuscules on obtient : `ord('A') = 65` , `ord('B') = 66` , ... , `ord('Z') = 90`

- La fonction `chr()` permet de faire l'opération inverse :

`chr(65) = 'A'` , `chr(66) = B` , ... , `chr(90) = 'Z'`

donc pour passer d'un caractère **majuscule** à un caractère **minuscule**, il suffit d'**ajouter 32** à son ordre.

Voici donc l'algorithme qui transforme une chaîne de caractères majuscules en une chaîne de caractères minuscules

```

1 # coding : utf-8
2
3 # fonction sui transforme une chaine en majuscule
4 def maj(s) :
5     # initialisation de la chaine
6     s_maj = ""
7     for x in s :
8         # on teste si x est un caractère minuscule
9         if ord(x) >= 97 and ord(x) <= 122 :
10             # on transforme x en majuscule
11             x = chr(ord(x) - 32)
12             s_maj = s_maj + x
13         else :
14             s_maj = s_maj + x
15     return s_maj
16
17 # Exemple
18 s1 = "hello world"
19 s2 = "Hello World !"
20 print(maj(s1)) # affiche HELLO WORLD
21 print(maj(s2)) # affiche HELLO WORLD !

```

Exercice 99. Écrire un algorithme sous forme de fonction en Python qui permet de transformer les chaînes de caractères en **minuscules** **sans utiliser la méthode `lower()`**.

Solution.

```

1 # coding : utf-8
2
3 # fonction sui transforme une chaine en minuscule
4 def min(s) :
5     # initialisation de la chaine en minuscule

```

```

6     s_min = ""
7     for x in s :
8         # on teste si x est un caractère majuscule
9         if ord(x) >= 65 and ord(x) <= 90 :
10            # on transforme x en minuscule
11            x = chr(ord(x) + 32)
12            s_min = s_min + x
13        else :
14            s_min = s_min + x
15    return s_min
16
17    # Exemple
18    s1 = "HELLO WORLD"
19    s2 = "Hello World !"
20    print(min(s1)) # affiche hello world
21    print(min(s2)) # affiche hello world !

```

Exercice 100. Écrire un algorithme en python sous forme de fonction Python permettant d'échanger la casse des caractères d'une chaîne sans utiliser les méthodes `upper()` , `lower()` , `swapcase()`...

Solution. Pour échanger la casse des caractères d'une chaîne, nous allons utiliser le code ASCII et les méthodes python associées (voir solution de l'exercice 98)

```

1  # coding : utf-8
2
3  # fonction qui échange la casse des caractères
4  def echangeCasse(s) :
5      # initialiser la chaine recherchée
6      echange_s = ""
7      # parcourir les caractères de la chaine s
8      for x in s :
9          # on teste si le caractère est en majuscule
10         if ord(x) >= 65 and ord(x) <= 90 :
11             x = chr(ord(x) + 32)
12             echange_s = echange_s + x
13         # on teste si le caractères est en minuscule
14         elif ord(x) >= 97 and ord(x) <= 122 :
15             x = chr(ord(x) - 32)
16             echange_s = echange_s + x
17         else :
18             echange_s = echange_s + x
19
20     return echange_s
21
22    # Exemple
23    s = "Hello World !"

```

```
24 print(exchangeCasse(s))
```

2. Exercices sur les listes Python

Exercice 101. Écrire un algorithme python pour afficher tous les éléments d'une liste donnée de deux manières différentes

Solution.

```
1 #coding : utf-8
2 # définir une liste python
3 L = ["laptop" , "ipad" , "iphone" , "tablette" , "
    imprimante"]
4
5 # première méthode
6 print("\npremière méthode\n—————")
7 for x in L :
8     print(x)
9
10 # deuxième méthode
11 print("\ndeuxième méthode\n—————")
12 n = len(L)
13 for i in range(0, n) :
14     print(L[i])
```

Exercice 102. Écrire un algorithme Python permettant d'échanger le premier élément avec le dernier élément d'une liste donnée. Exemple : si `L = ["Python", "Java", "C ++", "Javascript"]`, l'algorithme renvoie la liste : `["Javascript", "Java", "C ++", "Python"]`

Solution.

```
1 #coding : utf-8
2 def swapList(L) :
3     # obtenir le dernier élément de la liste
```

```

4     swap = L[-1]
5
6     # remplacer le dernier élément de la liste par le
    premier
7     L[-1] = L[0]
8
9     # remplacer le premier élément de la liste par le
    dernier
10    L[0] = swap
11    return L
12
13
14 # Exemple
15 L = L = ["Python" , "Java" , "C++" , "Javascript"]
16 print(swapList(L))
17 # La sortie est : ['Javascript', 'Java', 'C++', 'Python']

```

Exercice 103. Écrire un algorithme python sous forme de fonction qui prend en paramètres une liste `l` et qui renvoie un tuple de deux listes (`l_even`, `l_odd`) où `l_even` est composé des éléments de `l` d'indice pair et `l_odd` est constitué par les éléments d'indice impair. Exemple : si : `L = ["Python", "Java", "C ++", "C #", "VB.Net", "Javascript"]`, l'algorithme renvoie : `(['Python', 'C ++', 'VB.Net'], ['Java', 'C #', 'Javascript'])`

Solution.

```

1  def odd_event(L) :
2      # obtenir la longueur de la liste
3      n = len(L)
4      # initialisation des listes d'indices impair et d'
    indices pair
5      l_odd = []
6      l_even = []
7
8      # construire les liste l_odd et l_even
9      for i in range(0 , n) :
10         if( i%2 == 0) :
11             l_even.append(L[i])
12         else :
13             l_odd.append(L[i])
14
15     return (l_even , l_odd)
16
17 # Exemple
18 L = ["Python" , "Java" , "C++" , "C#" , "VB.Net" , "
    Javascript"]

```

```

19 print(odd_event(L))
20 # La sortie est : (['Python', 'C++', 'VB.Net'], ['Java',
    'C#', 'Javascript'])

```

Exercice 104. Écrire un programme en Python qui demande à l'utilisateur de saisir 5 nombres entiers de son choix et de lui afficher la liste des nombres saisis.

Solution.

```

1 # coding : utf-8
2 # initialisation de la liste des nombres à saisir
3 listNombres = []
4 for i in range(0,5) :
5     n = int(input("Tapez la valeur d'un entier : "))
6     # ajouter le nombre n à la liste
7     listNombres.append(n)
8 # Afficher la liste des nombres saisis :
9 print("Voici la liste des nombres saisis : " , listNombres
    )

```

Exercice 105. Etant donné une liste d'entiers L, écrire un programme en Python qui renvoie la somme des éléments de la liste L.

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui renvoie la somme des éléments
  d'une liste
3 def listSum(L) :
4     # initialisation de la somme des éléments de la liste
5     s = 0
6     for x in L :
7         s = s + x
8     return s
9
10 # Exemple
11 L = [2 , 1 , 5 , 3]
12 # Afficher la somme des éléments de la liste :
13 print("La somme des éléments de la liste L est : " ,
    listSum(L))
14 # La sortie est : La somme des éléments de la liste L est
    : 11

```

Exercice 106. Écrire un algorithme Python qui renvoie la **longueur** d'une **liste** donnée sans utiliser la méthode **len()**.

Solution.

```

1 # coding : utf-8
2 def lenList(L) :
3     # initialisation de la longueur de la liste
4     l = 0
5     # parcour les éléments de la liste
6     for x in L :
7         # incrémentation de la longueur
8         l = l + 1
9     return l
10
11 # Exemple
12 L = [11 , 3 , 5 , 8 , 2]
13 print("la longueur de la liste est : " , lenList(L))
14 # La sorite est : la longueur de la liste est : 5

```

Exercice 107. Écrire un algorithme Python qui renvoie la liste des diviseurs d'un entier donné. Exemple si $n = 18$, l'algorithme renvoie la liste [1, 2, 3, 6, 9, 18]

Solution.

```

1 # coding : utf-8
2 def listDiv(n) :
3     # initialisation de la liste des diviseur de n
4     l = []
5     # parcourt des entiers 1 , 2 , 3 , ..., n
6     for i in range(1, n+1) :
7         # si i est un diviseur de n on l'ajoute à la liste
8         if n%i == 0 :
9             l.append(i)
10    return l
11
12 # Exemple
13 n = 18
14 print("la liste des diviseurs de n est : " , listDiv(n))
15 # La sorite est : la liste des diviseurs de n est : [1,
    2, 3, 6, 9, 18]

```

Exercice 108. Écrire un algorithme en python qui renvoie la moyenne des termes d'une liste de nombres.

Solution.

```

1 #coding : utf-8
2 def moyList(L) :
3     # initialisation de la moyenne
4     m = 0
5     for x in L :
6         m = m + x
7     # moyenne somme des éléments de L / longueur de L
8     m = m/len(L)
9     return m
10
11 # Exemple
12 L = [3, 7, 8, 2]
13 print('la moyenne est : m = ', moyList(L))
14 # la sortie est : la moyenne est : m = 5.0

```

Exercice 109. Écrire un algorithme en python sous forme de fonction qui prends en paramètre un couple **(L, a)** formé d'une liste L et d'un élément a et qui renvoie **True** si l'élément a est présent dans la liste L et **False** si non.

Solution.

```

1 #coding : utf-8
2 def examList(L, a) :
3     if a in L :
4         return True
5     else :
6         return False
7
8 # Exemple
9 L = [13, 9, 5, 23]
10 a = 5
11 print(examList(L,a))
12 # la sortie est : True

```

Exercice 110. Écrire un algorithme en python sous forme de fonction qui prends en paramètre un couple (L, a) formé d'une liste L et d'un élément a et qui renvoie la position de l'élément a dans la liste L. La fonction doit renvoyer -1 si l'élément a n'est pas présent dans la liste.

Solution.

```

1 #coding : utf-8
2 def examinList(L, a) :
3     if a in L:
4         return L.index(a)
5     else :
6         return -1
7
8 # Exemple
9 L = [13, 9 , 5 , 23]
10 a = 9
11 b = 11
12 print(examinList(L,a)) # affiche 1
13 print(examinList(L,b)) # affiche -1

```

Exercice 111. Écrire un algorithme en python sous forme de fonction qui prends en paramètre un couple (L, n) formé d'une liste L de nombres et d'un entier n et qui renvoie la liste obtenue à partir de L en multipliant ses éléments par n. Exemple si $L = [3, 9, 5, 23]$ et $n = 2$ la fonction renvoie la liste : $[6, 18, 10, 46]$

Solution.

```

1 # coding : utf-8
2 def listMultiply(L,n) :
3     # initialisation de la liste souhaitée
4     l_mult = []
5     # parcourir et multiplier les éléments de la liste par
6     # n
7     for x in L :
8         l_mult.append(n*x)
9
10    return l_mult
11
12 # Exemple
13 n = 2
14 L = [3, 9 , 5 , 23]
15 print(listMultiply(L,n)) # affiche [6, 18, 10, 46]

```

Exercice 112. En utilisant la méthode `sort()`, écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste L de nombres et qui renvoie le couple (**min** , **max**) formé du minimum et du maximum de la liste.

Solution.

```

1 # coding : utf-8
2 def minMax(L) :
3     # trier la liste
4     L.sort()
5
6     return (L[0] , L[-1])
7
8 # Exemple
9 L = [41, 19 , 5 , 21 ,7 , 11]
10
11 print(minMax(L)) # affiche (5, 41)

```

Exercice 113. Écrire un algorithme python permettant de supprimer les éléments dupliqués d'une liste.

Solution.

```

1 # coding : utf-8
2 def removeDuplicate(L) :
3     # initialiser la liste sans éléments dupliqués
4     noDuplicate = []
5
6     # parcourir les éléments de L et supprimer les
7     # éléments dupliqués
8     for x in L :
9         if x not in noDuplicate :
10             noDuplicate.append(x)
11     return noDuplicate
12
13 # Exemple
14 L = [5 , 19 , 5 , 21 , 5 , 13 , 21, 5]
15 print("Liste sans éléments dupliqués : " , removeDuplicate(L))
16 # La sortie est : Liste sans éléments dupliqués : [5, 19, 21, 13]

```

Exercice 114. Écrire un programme Python sous forme de fonction Python qui prend en paramètres deux listes et renvoie True si les deux listes ont au moins un élément commun et False si non.

Solution.

```

1 # coding : utf-8
2 def commonElements(L1 , L2) :
3     # initialiser un compteur
4     compteur = 0
5
6     # parcourir les éléments de L1 et L2 et rechercher les
       éléments communs
7     for x in L1 :
8         if x in L2 :
9             compteur = compteur + 1
10    if compteur != 0 :
11        return True
12    else :
13        return False
14
15 # Exemple
16 L1 = [5 , 19 , 21 , 7 , 13 , 21]
17 L2 = [3 , 22 , 19 , 12 , 13 , 7]
18 L3 = [1 , 31 , 11 , 24 , 37]
19 print(commonElements(L1 , L2)) # affiche : True
20 print(commonElements(L1 , L3)) # affiche : False

```

Exercice 115. Écrire un programme Python sous forme de fonction Python qui prend en paramètres deux listes et renvoie la liste des éléments communs à ces deux listes.

Solution.

```

1 # coding : utf-8
2 def listCommonElements(L1 , L2) :
3     # initialisation de la liste des éléments communs
4     listCommon = []
5
6     # parcourir les éléments de L1 et L2 et rechercher les
       éléments communs
7     for x in L1 :
8         if x in L2 :
9             listCommon.append(x)
10    return listCommon
11
12 # Exemple
13 L1 = [5 , 19 , 21 , 7 , 13 , 21]
14 L2 = [3 , 22 , 19 , 12 , 13 , 7]
15 print(listCommonElements(L1 , L2)) # affiche : [19, 7, 13]

```

Exercice 116. Écrire un programme en python qui permet de mélanger au hasard les éléments d'une liste donnée.

Solution.

```

1 #coding : utf-8
2 from random import shuffle
3 laptop = [ 'HP', 'Acer', 'Dell', 'Lenovo', 'Thomson', 'Asus' ]
4 # mélanger les éléments de la liste avec le module shuffle
5 shuffle(laptop)
6 # afficher le résultat
7 print(laptop)

```

Exercice 117. Écrire un programme en Python permettant de réaliser la différence de deux liste. Exemple si :

$L_1 = [11, 3, 22, 7, 13, 23, 9]$ et $L_2 = [5, 9, 19, 23, 10, 23, 13]$, le programme renvoie la liste : $[11, 3, 22, 7]$

Solution.

```

1 #coding : utf-8
2 def differenceList(L1 , L2) :
3     # initialiser la liste difference de L1 et L2
4     diff = []
5     for x in L1 :
6         if x not in L2 :
7             diff.append(x)
8     return diff
9 #Exemple
10 L1 = [11 , 3 , 22 , 7 , 13 , 23 , 9]
11 L2 = [5 , 9 , 19 , 23 , 10 , 23 , 13]
12 print("La différence de L1 et L2 est : " , differenceList(
13     L1 , L2))
13 # La sortie est : La différence de L1 et L2 est : [11, 3,
    22, 7]

```

Exercice 118. Écrire un programme en Python permettant de réaliser la différence symétrique de deux listes L1 et L2, c.a.d la liste formée des éléments de L1 qui ne sont pas dans L2 et les éléments de L2 qui ne sont pas dans L1 Exemple si :

$L_1 = [11, 3, 22, 7, 13, 23, 9]$ et $L_2 = [5, 9, 19, 23, 22, 23, 13]$, le programme renvoie la liste : $[11, 3, 7, 5, 19]$

Solution.

```

1 #coding : utf-8
2 def differenceSymetrique(L1 , L2) :
3     # initialiser la liste difference symétrique de L1 et
      L2
4     diffSym = []
5     for x in L1 :
6         if x not in L2 :
7             diffSym.append(x)
8     for x in L2 :
9         if x not in L1 :
10            diffSym.append(x)
11    return diffSym
12 #Exemple
13 L1 = [11 , 3 , 22 , 7 , 13 , 23 , 9]
14 L2 = [5 , 9 , 19 , 23 , 22 , 23 , 13]
15 print("La différence symétrique de L1 et L2 est : " ,
      differenceSymetrique(L1 , L2))
16 # La différence symétrique de L1 et L2 est : [11, 3, 7,
      5, 19]

```

Exercice 119. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste de nombres L et qui renvoie le minimum de la liste sans utiliser aucune fonction prédéfinie en Python.

Solution.

```

1 #coding : utf-8
2 def minimumList(L) :
3     # initialiser le minimum de la liste
4     minList = L[0]
5     for x in L :
6         if minList > x :
7             minList = x
8
9     return minList
10 #Exemple
11 L = [15 , 31 , 12 , 7 , 19 , 23 , 29]
12
13 print("Le minimum de la liste L est : " , minimumList(L))
14 # La sortie est : Le minimum de la liste L est : 7

```

Exercice 120. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste de nombres L et qui renvoie le maximum de la liste sans utiliser aucune fonction prédéfinie en Python.

Solution.

```

1 #coding: utf-8
2 def maximumList(L) :
3     # initialiser le minimum de la liste
4     maxList = L[0]
5     for x in L :
6         if maxList < x :
7             maxList = x
8
9     return maxList
10 #Exemple
11 L = [15 , 31 , 12 , 7 , 19 , 23 , 29]
12
13 print("Le maximum de la liste L est : " , maximumList(L))
14 # La sortie est : le maximum de la liste L est : 31

```

Exercice 121. Écrire un algorithme en python qui renvoie la liste des éléments dupliqués d'une liste données. Exemple si $L = [7, 23, 5, 12, 7, 19, 23, 12, 29]$, l'algorithme renvoie la liste : $[7, 23, 12]$

Solution.

```

1 #coding: utf-8
2 def listDuplicate(L) :
3     # initialiser la liste des éléments dupliqués
4     duplicate = []
5     for x in L :
6         if L.count(x) > 1 and x not in duplicate :
7             duplicate.append(x)
8     return duplicate
9 #Exemple
10 L = [7 , 23 , 5 , 12 , 7 , 19 , 23 , 12 , 29]
11
12 print("la liste des éléments dupliqués de L est : " ,
13       listDuplicate(L))
14 # La sortie est : la liste des éléments dupliqués de L est
15 : [7, 23, 12]

```

Exercice 122. Écrire un algorithme en python qui renvoie la liste des éléments dupliqués d'une liste données sans utiliser aucune fonction prédéfinie en Python. Exemple si $L = [7, 23, 5, 12, 7, 19, 23, 12, 29]$, l'algorithme renvoie la liste $[7, 23, 12]$.

Solution.

```

1 #coding : utf-8
2 def listDuplicate(L) :
3     # initialisation de la liste des éléments dupliqués
4     duplicateElements = []
5
6     for x in L :
7         # initialiser l'occurrence de x dans L
8         occ_x = 0
9         for y in L :
10             if x == y :
11                 occ_x = occ_x + 1
12
13         # tester si x est un élément dupliqué et ajouter
14         # le à la liste duplicateElements
15         if occ_x >= 2 and x not in duplicateElements :
16             duplicateElements.append(x)
17
18     return duplicateElements
19
20 #Exemple
21 L = [7 , 23 , 5 , 12 , 7 , 19 , 23 , 12 , 29]
22 print(listDuplicate(L)) # la sortie est : [7, 23, 12]

```

Exercice 123. Écrire un algorithme en python qui renvoie le nombre d'occurrences d'un élément a dans une liste L donnée sans utiliser aucune fonction prédéfinie en Python. Exemple si $L = [7, 23, 5, 23, 7, 19, 23, 12, 29]$ et $a = 23$, l'algorithme renvoie 3.

Solution.

```

1 #coding : utf-8
2 def numberOccurrence(L , a) :
3     # initialiser le nombre d'occurrence de a dans L
4     numberOcc = 0
5     for x in L :
6         if x == a :
7             numberOcc = numberOcc + 1
8     return numberOcc
9
10 #Exemple
11 L = [7 , 23 , 5 , 23 , 7 , 19 , 23 , 12 , 29]
12 a = 23
13 print("le nombre d'occurrences de a dans L est : " ,
14       numberOccurrence(L , a))
15 # La sortie est : le nombre d'occurrences de a dans L est
16 : 3

```


Exercice 124. Écrire un algorithme en python sous forme de fonction qui prends en paramètre un couple (L, a) formé d'une liste L et d'un élément a et qui renvoie la position de l'élément a dans la liste L sans utiliser la fonction index() ni aucune autre fonction prédéfinie en Python. La fonction doit renvoyer -1 si l'élément a n'est pas présent dans la liste.

Solution.

```

1 # coding : utf-8
2 def indexElement(L , a) :
3     # initialisation de l'index de a
4     index = -1
5     for i in range(0 , len(L)) :
6         if L[i] == a :
7             index = i
8             break
9     return index
10
11 # Exemple
12 L = [2 , 11 , 7 , 4 , 3 , 7 , 13]
13 a = 7
14 print("l'index de a dans L est : " , indexElement(L , 7))
15     # affiche : l'index de a dans L est : 2
16 print("l'index de a dans L est : " , indexElement(L , 5))
17     # affiche : l'index de a dans L est : -1

```

Exercice 125. Écrire un programme en Python qui détermine la liste des mots ne contenant pas la lettre 'a' dans un texte donné.

Solution.

```

1 #coding : utf-8
2 def motSans_a(T) :
3     #initialiser la liste des mots qui ne contiennent pas
4     la lettre a
5     listMotSans_a = []
6     # convertir le texte T en une liste
7     L = T.split()
8     # parcourir les mots de la liste L et rechercher ceux
9     qui ne contiennent pas la lettre a
10    for mot in L :
11        if 'a' not in mot :
12            listMotSans_a.append(mot)
13    return listMotSans_a
14
15 # Exemple
16 T = "Python est un langage de programmation de haut niveau"

```

```

15 print("la liste des mots qui ne contiennent pas a est :",
        motSans_a(T))
16 # La sortie est : la liste des mots qui ne contiennent pas
    a est : ['Python', 'est', 'un', 'de', 'de']

```

Exercice 126. Etant donné une liste de nombres entiers L, écrire un algorithme en Python permettant de renvoyer le couple de listes (**l_pair**, **l_impair**) où **l_pair** désigne la liste des entiers pairs de L et **l_impair** désigne la listes des entiers impairs de L. Exemple pour L = [11, 3, 22, 14, 31, 18, 12, 7], le programme renvoie le couple de listes : ([22, 14, 18, 12], [11, 3, 31, 7])

Solution.

```

1 def pairImpair(L) :
2     # initialisation des listes d'entiers pair et impair
3     l_pair = []
4     l_impair = []
5     for n in L :
6         if n%2 == 0 :
7             l_pair.append(n)
8         else :
9             l_impair.append(n)
10    return (l_pair, l_impair)
11
12 #Exemple
13 L = [11, 3, 22, 14, 31, 18, 12, 7]
14 print(pairImpair(L)) # affiche ([22, 14, 18, 12], [11, 3,
    31, 7])

```

Exercice 127. Écrire un algorithme Python qui détermine la liste des mots commençant par une majuscule dans un texte T donné.

Solution.

```

1 #coding: utf-8
2 def beginMaj( T ) :
3     # initialisation de la liste des mots qui commencent
    par une majuscule
4     maj = []
5     # convertir le texte T en une liste :
6     L = T.split()
7     for mot in L :
8         if (mot[0].isupper()) :

```

```

9         maj.append(mot)
10     return maj
11 # Exemple
12 T = "Python is more power thant Java"
13 print(beginMaj(T)) # affiche ['Python', 'Java']

```

Exercice 128. Écrire un algorithme Python qui détermine la liste des mots contenant au moins deux majuscule dans un texte T donné.

Solution.

```

1 # coding : utf-8
2 T = "Python est un langage de programmation qui prends en
   charge mySql, mongoDB, postgreSQL"
3
4 # fonction qui teste si un mot contient au moins 2
   majuscules
5 def contains2Majuscule(mot) :
6     # initialiser un compteur
7     compteur = 0
8     for x in mot :
9         if (x.isupper()) :
10             compteur = compteur + 1
11     if compteur < 2 :
12         return False
13     else :
14         return True
15
16 # fonction qui détermine la liste des mots contenant au
   moins une majuscule
17 def list2Maj(T) :
18     # liste des mots ayant au moins deux majuscules
19     list2Maj = []
20     # convertir T en une liste
21     L = T.split()
22     for word in L :
23         if contains2Majuscule(word) :
24             list2Maj.append(word)
25     return list2Maj
26
27 print("La liste des mots ayant au moins 2 majuscules est :
   ", list2Maj(T))

```

Exercice 129. Écrire un algorithme Python qui détermine la liste des mots contenant au moins un chiffre dans un texte T donné.

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui detecte si un mot contient
  un chiffre ou non
3 def digitInWord(mot) :
4     compteur = 0
5     for x in mot :
6         if x.isdigit() :
7             compteur = compteur + 1
8     if compteur > 0 :
9         return True
10    else :
11        return False
12
13 # création d'une fonction qui renvoie la liste des mots
  contenant au moins un chiffre
14 def digitInText(T) :
15     # initialiser la liste des mots contenant au moins un
  chiffre
16     listMotAvecChiffre = []
17     # convertir le texte T en une liste
18     L = T.split()
19     for mot in L :
20         # on test si le mot 'mot' contient un chiffre
21         # à l'aide de la fonction digitInWord définit
  précédemment
22         if digitInWord(mot) :
23             listMotAvecChiffre.append(mot)
24     return listMotAvecChiffre
25 # Exemple
26 T = "Python2.7 est remplacé par Python3.X depuis depuis
  2018"
27 print("la liste des mots contenant au moins un chiffre est
  \n" , digitInText(T))
28 # renvoie la liste : ['Python2.7', 'Python3.X', '2018']

```

Exercice 130. Écrire un algorithme Python qui détermine la liste des mots ne contenant aucun chiffre dans un texte T donné.

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui detecte si un mot contient
  un chiffre ou non
3 def digitInWord(mot) :
4     compteur = 0
5     for x in mot :
6         if x.isdigit() :

```

```

7         compteur = compteur + 1
8     if compteur > 0 :
9         return True
10    else :
11        return False
12
13    # création d'une fonction qui renvoie la liste des mots ne
        contenant aucun chiffre
14    def noDigitInText(T) :
15        # initialiser la liste des mots ne contenant aucun
            chiffre
16        listMotSansChiffre = []
17        # convertir le texte T en une liste
18        L = T.split()
19        for mot in L :
20            # on test si le mot 'mot' ne contient aucun
                chiffre
21            # à l'aide de la fonction digitInWord définit
                précédemment
22                if not digitInWord(mot) :
23                    listMotSansChiffre.append(mot)
24        return listMotSansChiffre
25    # Exemple
26    T = "Python2.7 est remplacé par Python3.X depuis depuis
        2018"
27    print("la liste des mots ne contenant aucun est\n" ,
        noDigitInText(T))
28    # renvoie la liste : ['est', 'remplacé', 'par', 'depuis',
        'depuis']

```

Exercice 131. Etant donné une liste de nombres entiers L , écrire un algorithme en Python permettant de renvoyer la liste des couples (n, m) vérifiant $n + m < 10$ avec m et n appartiennent à L . Exemple si $L = [11, 3, 2, 22, 4, 31, 18, 6, 12, 1, 7]$, le programme renvoie : la liste : $[(3, 3), (2, 3), (4, 3), (6, 3), (1, 3), (3, 2), (2, 2), (4, 2), (6, 2), (1, 2), (7, 2), (3, 4), (2, 4), (4, 4), (1, 4), (3, 6), (2, 6), (1, 6), (3, 1), (2, 1), (4, 1), (6, 1), (1, 1), (7, 1), (2, 7), (1, 7)]$

Solution.

```

1
2    # coding : utf-8
3    def listTuples(L) :
4        # initialisation de la liste des tuples
5        lTuples = []
6        for n in L :
7            for m in L :
8                if m + n < 10 :

```

```

9         lTuples.append((m,n))
10    return lTuples
11    # Exemple
12    L = [11 , 3 , 2 , 22 , 4 , 31 , 18 , 6, 12 , 1 , 7]
13    print(listTuples(L))
14    # affiche :
15    """
16    [(3, 3), (2, 3), (4, 3), (6, 3), (1, 3),
17     (3, 2), (2, 2), (4, 2), (6, 2), (1, 2),
18     (7, 2), (3, 4), (2, 4), (4, 4), (1, 4),
19     (3, 6), (2, 6), (1, 6), (3, 1), (2, 1),
20     (4, 1), (6, 1), (1, 1), (7, 1), (2, 7),
21     (1, 7)]
22    """

```

Exercice 132. Écrire un programme Python qui permet de déplacer les 3 premiers élément d’une liste donnée et les placer à la fin de la liste. Exemple si $L = [25, 13, 11, 1, 4, 31, 18, 6, 12, 1, 7]$, le programme renvoie : la liste : $[1, 4, 31, 18, 6, 12, 1, 7, 25, 13, 11]$

Solution.

```

1  # coding : utf-8
2  def moveListElements(L) :
3      # suppression des trois élément de la liste
4      L1 = L[3 : :]
5      # extraction des trois premier éléments
6      L2 = L[0 :3]
7
8      return L1 + L2
9
10 L = [25 , 13 , 11 , 1 , 4 , 31 , 18 , 6, 12 , 1 , 7]
11 print(moveListElements(L))
12 # Affiche : [1, 4, 31, 18, 6, 12, 1, 7, 25, 13, 11]

```

Exercice 133. Écrire un programme Python sous forme de fonction qui prend en paramètre une liste L et renvoie la liste obtenue en effectuant une permutation circulaire sur la liste L. Exemple si $L = [41, 11, 34, 20, 18, 6]$, le programme renvoie : la liste : $[6, 41, 11, 34, 20, 18]$

Solution.

```

1  # coding : utf-8
2  def circularPermutation(L) :

```

```

3     # initialisation de la liste obtenue via une
      permutation circulaire de L
4     circularList = []
5     for i in range(1, len(L)) :
6         circularList.append(L[i-1])
7     circularList.insert(0,L[-1])
8     return circularList
9
10
11 L = [41 , 11 , 34 , 20 , 18 , 6]
12 print(circularPermutation(L))
13 # Affiche : [6, 41, 11, 34, 20, 18]

```

Exercice 134. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une chaîne de caractères texte T et qui renvoie la liste des mots contenant au moins deux chiffres. Exemple : si **T = 'Python2.7 est remplacé par Python3.X depuis depuis 2018'**, la fonction renvoie la liste ['Python2.7' , '2018']

Solution.

```

1  # coding : utf-8
2  # création d'une fonction qui calcul le nombre de chiffre
      sur une chaîne
3  def number_digit(s) :
4      # initialisation du nombre de chiffres
5      nbrDigit = 0
6      for x in s :
7          if x.isdigit() :
8              nbrDigit = nbrDigit + 1
9      return nbrDigit
10
11 # création de la fonction qui renvoie la liste des mots
      contenant au moins deux chiffres
12 def list2Digits(s) :
13     # initialisation de la liste des mots recherchée
14     listMot = []
15     # convertir la chaîne s en une liste
16     L = s.split()
17     for mot in L :
18         if number_digit(mot) >= 2 :
19             listMot.append(mot)
20     return listMot
21
22 # Exemple
23 T = "Python2.7 est remplacé par Python3.X depuis depuis
      2018"
24 print(list2Digits(T)) # affiche : ['Python2.7', '2018']

```

Exercice 135. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une chaîne texte T et qui renvoie la liste des mots contenant au moins un chiffre et une majuscule. Exemple : si **T = 'Python2.7 est remplacé par Python3.X depuis 2018'**, la fonction renvoie la liste : ['Python2.7' , 'Python3.X']

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui calcul le nombre de chiffre
  sur une chaîne
3 def number_digit(s) :
4     # initialisation du nombre de chiffres
5     nbrDigit = 0
6     for x in s :
7         if x.isdigit() :
8             nbrDigit = nbrDigit + 1
9     return nbrDigit
10
11 # création d'une fonction qui calcul le nombre de
    majuscule dans une chaîne
12 def number_maj(s) :
13     # initialisation du nombre de majuscules
14     nbrMaj = 0
15     for x in s :
16         if x.isupper() :
17             nbrMaj = nbrMaj + 1
18     return nbrMaj
19
20 # création de la fonction qui renvoie la liste des mots
    contenant au moins un chiffre et une majuscule
21 def listMajDigits(s) :
22     # initialisation de la liste des mots recherchée
23     listMot = []
24     # convertir la chaîne s en une liste
25     L = s.split()
26     for mot in L :
27         if number_digit(mot) > 0 and number_maj(mot) > 0 :
28             listMot.append(mot)
29     return listMot
30
31 # Exemple
32 T = "Python2.7 est remplacé par Python3.X depuis 2018"
33 print(listMajDigits(T)) # affiche : ['Python2.7' , 'Python3.
    X']

```

Exercice 136. Écrire un algorithme en python sous forme de fonction qui prends en paramètre un couple (**listeNotes** , **listeCoefficients**) et qui

renvoie la moyenne obtenue, avec **listeNotes** désigne la liste des notes obtenues par un étudiant et **listeCoefficients** désigne la liste des coefficients associés.

Solution.

```

1 # coding : utf-8
2 def average(listeNotes , listeCoefficients) :
3     # initialisation de la moyenne
4     average = 0
5     # somme des coefficients initialisé
6     sumCoefficients = 0
7
8     for i in range(0 , len(listeNotes)) :
9         # calculer la somme des coefficients*notes
10        average = average + listeNotes[i] *
        listeCoefficients[i]
11        # calculer la somme des coefficients
12        sumCoefficients = sumCoefficients +
        listeCoefficients[i]
13        average = average/sumCoefficients
14    return average
15
16 # Exemple
17 listeNotes = [16 , 14 , 12 ]
18 listeCoefficients = [2 , 1 , 4 ]
19 print("La moyenne est : " , average(listeNotes ,
        listeCoefficients))
20 # La sortie est : La moyenne est : 13.428571428571429

```

Exercice 137. Écrire un algorithme en python qui permet d'extraire d'une liste de nombres la sous liste formée des nombres qui contiennent le chiffre 3.

Solution.

```

1 # coding : utf-8
2 # création d'une méthode qui test si un nombre contient ou
    non le chiffre 3
3 def contains3(n) :
4     # convertir n en string
5     s = str(n)
6     if '3' in s :
7         return True
8     else :
9         return False
10
11 # méthode qui détermine la liste des nombres qui
    contiennent le chiffre 3

```

```

12
13 def listContains3(L) :
14     # initialisation de la liste des nombres qui
        contiennent le chiffre 3
15     list3 = []
16     for n in L :
17         if contains3(n) :
18             list3.append(n)
19     return list3
20
21 # Exemple
22 L = [ 21 , 137, 25 , 31 , 71 , 239]
23 print(listContains3(L)) # affiche [137, 31, 239]

```

Exercice 138. Écrire un algorithme en python qui permet d'extraire d'une liste de nombres la sous liste formée des nombres qui contiennent le chiffre 3 et ne contiennent pas le chiffre 2.

Solution.

```

1 # coding : utf-8
2 # fonction qui test si un nombre contient le chiffre 3 et
        ne contient pas 2
3 def contains(n) :
4     # convertir n en string
5     s = str(n)
6     if '3' in s and '2' not in s :
7         return True
8     else :
9         return False
10
11 # fonction qui renvoie la liste des nombres qui
        contiennent le chiffre 3 et ne contiennent pas 2
12
13 def listContains(L) :
14     # initialisation de la liste des nombres qui
        contiennent le chiffre 3
15     list3 = []
16     for n in L :
17         if contains(n) :
18             list3.append(n)
19     return list3
20
21 # Exemple
22 L = [ 21 , 137, 25 , 31 , 71 , 239]
23 print(listContains(L)) # affiche [137, 31]

```

Exercice 139. Ecrire un algorithme en python qui permet d'extraire d'une chaîne de caractères s, la liste des mots contenant au moins deux majuscules successives. Exemple si s = "SQLite et MongoDB sont beaucoup plus simple que les bases de données PostgreSQL et Oracle" , l'algorithme renvoie la liste L = [SQLite , MongoDB , PostgreSQL]

Solution.

```

1 # coding : utf-8
2
3 # fonction qui determine la liste des mots contenant au
  moins deux majuscules successives
4 def twoMajConsecutive(s) :
5     # initialisation de la liste recherchée
6     L_2_maj = []
7
8     # convertir la chaîne de caractère s en une liste
9     list_word = s.split()
10
11    # parcourir les mots de la liste
12    for word in list_word :
13        # rechercher les majuscules successives
14        for i in range(0 , len(word)-1) :
15            if word[i:i+2].isupper() and word not in
16                L_2_maj :
17                L_2_maj.append(word)
18
19    return L_2_maj
20
21 # Exemple
22 s = "SQLite et MongoDB sont beaucoup plus simple que les
   bases de données PostgreSQL et Oracle"
23 print(twoMajConsecutive(s))
24 # le programme affiche : ['SQLite', 'MongoDB', 'PostgreSQL']

```

Exercice 140. Écrire un algorithme en python qui permet d'extraire d'une liste de nombres la sous liste des nombres qui se terminent par un chiffre pair. Exemple : si L = [21 , 14 , 346 , 728 , 13 , 19], l'algorithme renvoie la liste : [14, 346, 728]

Solution.

```

1 # coding : utf-8
2 def listEven(L) :
3     # initialisation de la liste des nombres qui se
   terminent par un chiffre pair

```

```

4     lEvent = []
5     for n in L:
6         # déterminer le dernier chiffre du nombre n
7         last = n%10
8         # tester si le dernier chiffre est pair
9         if last % 2 == 0:
10            lEvent.append(n)
11     return lEvent
12
13 # Exemple
14 L = [21 , 14 , 346 , 728 , 13 , 19]
15 print(listEven(L)) # affiche [14, 346, 728]

```

Exercice 141. Écrire un algorithme en python qui permet d'extraire d'une liste de nombres la sous liste des nombres dont le dernier chiffre est pair et l'avant dernier est impair. Exemple si : $L = [21, 14, 346, 728, 136, 19]$, l'algorithme renvoie la liste : $[14, 136]$

Solution.

```

1
2 # coding : utf-8
3 def listEvenOdd(L) :
4     # initialisation de la liste des nombres dont le
5     # dernier chiffre pair et le suivant impair
6     lEventOdd = []
7     for n in L:
8         # déterminer le dernier chiffre du nombre n
9         last = n%10
10        second = ((n- last)//10)%10
11        # tester si le dernier chiffre est pair et le
12        # second impair
13        if last % 2 == 0 and second%2 != 0 :
14            lEventOdd.append(n)
15    return lEventOdd
16
17 # Exemple
18 L = [21 , 14 , 346 , 728 , 136 , 19]
19 print(listEvenOdd(L)) # affiche [14, 136]

```

Exercice 142. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste de nombres entiers L et qui renvoie la liste obtenue à partir de L en insérant juste après chaque nombre la chaîne de caractères 'pair' ou 'impair' selon la parité de nombre. Exemple si $L = [2$

, 11 , 25 , 6 , 14], l'algorithme renvoie la liste : [2 , 'pair' , 11 , 'impair' , 25 , 'impair' , 6 , 'pair' , 14 , 'pair']

Solution.

```

1 # coding : utf-8
2 def listEvenOdd(L) :
3     # initialisation de la liste cherchée
4     lEventOdd = []
5     for n in L :
6         # On ajoute l'élément n à la liste
7         lEventOdd.append(n)
8         if n%2 == 0 :
9             lEventOdd.append( 'pair' )
10        else :
11            lEventOdd.append( 'impair' )
12
13    return lEventOdd
14
15 # Exemple
16 L = [21 , 12 , 54 , 71 , 13 , 20]
17 print( listEvenOdd(L) )
18 # [21, 'impair', 12, 'pair', 54, 'pair', 71, 'impair', 13,
    'impair', 20, 'pair']

```

Exercice 143. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste de nombres entiers L et qui renvoie la liste obtenue à partir de L en supprimant tous les nombres négatifs. Exemple si L = [7 , -2 , 11 , -25 , 16 , -3 , 14], l'algorithme renvoie la liste : [7 , 11 , 16 , 14]

Solution.

```

1 #coding : utf-8
2 def deleteNegative(L) :
3     for x in L :
4         if x < 0 :
5             L.remove(x)
6     return L
7
8 L = [7 , -2 , 11 , -25 , 16 , -3 , 14]
9 print(deleteNegative (L) ) # affiche [7, 11, 16, 14]

```

Exercice 144. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste de nombres entiers L et qui renvoie la

liste obtenue à partir de L en déplaçant tous les zéros au début de la liste. Exemple si $L = [7, 0, 11, 0, 25, 16, 0, 14]$, l'algorithme renvoie la liste : $[0, 0, 0, 7, 11, 25, 16, 14]$

Solution.

```

1 # coding : utf-8
2 def move(L) :
3     # liste extraite de L formée uniquement des zéros de L
4     l_zero = []
5     # liste extraite de L formée uniquement des éléments
        non nuls de L
6     l_without_zero = []
7     for x in L :
8         if x == 0 :
9             l_zero.append(x)
10        else :
11            l_without_zero.append(x)
12    return l_zero + l_without_zero
13
14 # Exemple :
15 L = [7 , 0 , 11 , 0 , 25 , 16 , 0 , 14]
16 print(move(L)) # affiche : [0, 0, 0, 7, 11, 25, 16, 14]
```

Exercice 145. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste de nombres réels L et qui renvoie la liste obtenue à partir de L en supprimant tous les nombres entiers.

Solution.

```

1 # coding : utf-8
2 # Fonction qui supprime les entiers d'une liste
3 def removeInt(L) :
4     # initialisation de la liste obtenue à partir de L en
        supprimant tous les nombres entiers
5     l_remove_int = []
6     # liste extraite de L formée uniquement des nombres
        non entiers de L
7     for x in L :
8         if (type(x) != int) :
9             l_remove_int.append(x)
10    return l_remove_int
11
12 # Exemple :
13 L = [11.5 , 0 , 7.75 , 8 , 23.97 , 16 , 10 , 14.5]
14 print(removeInt(L)) # affiche : [11.5, 7.75, 23.97, 14.5]
```

Exercice 146. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une variable chaîne de caractères s et qui renvoie la liste de tous les caractères en majuscules qui se trouvent dans s.

Solution.

```

1 # coding : utf-8
2 def listMaj(s) :
3     # initialisation de la liste des majuscules
4     l_maj = []
5
6     for x in s :
7         if x.isupper() :
8             l_maj.append(x)
9     return l_maj
10
11 # Exemple :
12 s = "Tkinter is the most popular GUI Python Framework"
13 print(listMaj(s)) # affiche : ['T', 'G', 'U', 'I', 'P', 'F']

```

Exercice 147. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une variable chaîne de caractères s et qui renvoie la liste de tous les chiffres qui se trouvent dans s.

Solution.

```

1 # coding : utf-8
2 def listDigit(s) :
3     # initialisation de la liste des chiffres contenu dans
4     s
5     l_digit = []
6
7     for x in s :
8         if x.isdigit() :
9             l_digit.append(x)
10    return l_digit
11
12 # Exemple :
13 s = "La date de sortie de Python3.7.2 est le 03 décembre
14    2018"
15 print(listDigit(s)) # affiche : ['3', '7', '2', '0', '3',
16    '2', '0', '1', '8']

```

Exercice 148. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste L et qui renvoie la somme des élément d'index impair. Exemple si $L = [3, 2, 5, 11, 21, 4, 7]$, l'algorithme renvoie le nombre 17.

Solution.

```

1 #coding : utf-8
2 def oddSum(L) :
3     n = len(L)
4     # initialisation de la somme des éléments d'index
    impair
5     s = 0
6     for i in range(0,n) :
7         if i%2 != 0 :
8             s = s + L[i]
9     return s
10 # Exemple
11 L = [3 , 2 , 5 , 11 , 21 , 4 , 7]
12 print(oddSum(L)) # affiche 17

```

Exercice 149. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste L et qui renvoie le maximum des éléments d'index pair sans utiliser aucune fonction prédéfinie en Python. Exemple si $L = [13, 2, 31, 120, 4, 97, 15]$, l'algorithme renvoie le nombre 31.

Solution.

```

1 #coding : utf-8
2 def maxEven(L) :
3     # initialisation du maximum des éléments d'index pair
    evenMax = L[0]
4     # parcourir les éléments d'index pair de la liste
5     for i in range(0 , len(L)) :
6         if i%2 == 0 :
7             # chaque fois que evenMax < L[i] remplacer
            evenMax par L[i]
8             if evenMax < L[i] :
9                 evenMax = L[i]
10    return evenMax
11
12
13 # Exemple
14 L = [13 , 2 , 31 , 120 , 4 , 97 , 15]
15 print(maxEven(L)) # affiche 31

```


Exercice 150. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste L et qui renvoie la liste des éléments dupliqués au moins 3 fois.

Solution.

```

1 #coding : utf-8
2 # fonction qui détermine les éléments dupliqués au moins
  3 fois
3 def duplicate3(L) :
4     # initialisation de la liste des éléments dupliqués
      au moins 3 fois
5     dup3 = []
6     for i in range(0,len(L)) :
7         if L.count(L[i]) >= 3 and L[i] not in dup3 :
8             dup3.append(L[i])
9     return dup3
10 # Exemple
11 L = [3 , 7 , 3 , 7 , 12 , 5 , 7 , 12 , 31 , 12 , 3]
12 print(duplicate3(L)) # affiche : [3, 7, 12]
```

Exercice 151. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste L et qui renvoie la liste des éléments **dupliqués au moins 3 fois** sans utiliser la méthode count ni aucune méthode prédéfinie en Python. (Indication : utiliser l'exercice 123)

Solution.

```

1 #coding : utf-8
2 # fonction qui détermine le nombre d'occurrences d'un
  caractère dans une liste
3 def numberOccurrence(L , a) :
4     # initialiser le nombre d'occurrence de a dans L
5     numberOcc = 0
6     for x in L :
7         if x == a :
8             numberOcc = numberOcc + 1
9     return numberOcc
10
11 # fonction qui détermine la liste des élément dupliqués au
  moins 3 fois
12 def duplicate3(L) :
13     #initialisation de la liste des élément dupliqués
14     ldup = []
15     for x in L :
16         if(numberOccurrence(L , x) >= 3 and x not in ldup)
17         :
            ldup.append(x)
```

```

18     return ldup
19
20 #Exemple
21 L = [5 , 7 , 23 , 5 , 23 , 7 , 5 , 19 , 23 , 4 , 7 , 29 ,
      7]
22 print(duplicate3(L)) # affiche : [5, 7, 23]

```

Exercice 152. Écrire un programme Python sous forme de fonction qui prends en paramètres un couple (L , a) formé d’une liste L et un élément a et qui renvoie la liste des index de a dans la liste L. Exemple si L = [2 , 7 , 11 , 7 , 21 , 39 , 7] et a = 7 la fonction renvoie [1, 3, 6]

Solution.

```

1 #coding : utf-8
2 def listIndex(L , a) :
3     # initialisation de la liste des index
4     lIndex = []
5     for i in range(0, len(L)) :
6         if L[i] == a :
7             lIndex.append(i)
8     return lIndex
9
10 # Exemple
11 L = [2 , 7 , 11 , 7 , 21 , 39 , 7]
12 a = 7
13 print(listIndex(L , a)) # affiche [1, 3, 6]

```

Exercice 153. Écrire un algorithme en Python permettant de déterminer l’avant dernier index d’un élément dans une liste sans utiliser aucune fonction prédéfinie en Python. Exemple si L = [2 , 7 , 11 , 7 , 21 , 39 , 7] et a = 7 l’algorithme renvoie 3.

Solution.

```

1 #coding : utf-8
2 # fonction qui détermine l’index de l’avant dernier
  élément dans une liste
3 def penultimate(L , a) :
4     # initialisation de la liste des index
5     lIndex = []
6     for i in range(0, len(L)) :
7         if L[i] == a :
8             lIndex.append(i)

```

```

9     return lIndex[ len(lIndex) - 2 ]
10
11 # Exemple
12 L = [ 2 , 7 , 11 , 7 , 21 , 39 , 7 ]
13 a = 7
14 print( penultimate( L , a ) ) # affiche 3

```

Exercice 154. Écrire un algorithme en python sous forme de fonction qui prends en paramètre une liste d'entier $L = [n_1, n_2, n_3, \dots, n_p]$ et qui renvoie la liste : $[n_1, n_1 + n_2, n_1 + n_2 + n_3, \dots, n_1 + n_2 + \dots + n_p]$

Solution.

```

1 #coding : utf-8
2 def transformedList(L) :
3     # initialisation de la liste transformée
4     lTransformed = []
5     # initialisation des items
6     item = 0
7     for i in range(0, len(L)) :
8         item = item + L[i]
9         lTransformed.append(item)
10    return lTransformed
11
12 # Exemple
13 L = [ 2 , 1 , 5 , 3 , 4 ]
14 print( transformedList(L) ) # affiche [2, 3, 8, 11, 15]

```

Exercice 155. Écrire un algorithme en Python permettant de trier une liste selon l'algorithme du tri par **insertion**.

Solution.

```

1 def sort(L) :
2
3     for i in range(1, len(L)) :
4         current = L[i]
5         for j in range(i - 1, -1, -1) :
6             if L[j] > current :
7                 L[j], L[j + 1] = L[j + 1],
8                 L[j]
9             else :
10                L[j + 1] = current
11                break
12
13    return L

```

```

12
13 # Exemple
14 L = [41 , 23 , 7 , 31 , 27 , 5 , 2 , 17 , 25 ]
15 print(sort(L))
16 # affiche : [2, 5, 7, 17, 23, 25, 27, 31, 41]

```

Exercice 156. Écrire un algorithme en Python permettant de remplacer les éléments d'index pair d'une liste par 0 et ceux d'index impair par 1 .

Solution.

```

1 facile laissé à la sagacité du lecteur

```

Exercice 157. Écrire un algorithme en Python permettant de trier une liste selon l'algorithme du tri par **insertion**.

Solution.

```

1 def sort(L) :
2
3     for i in range(1, len(L)) :
4         current = L[i]
5         for j in range(i - 1, -1, -1) :
6             if L[j] > current :
7                 L[j], L[j + 1] = L[j + 1],
8                 L[j]
9             else :
10                L[j + 1] = current
11                break
12
13     return L
14
15 # Exemple
16 L = [41 , 23 , 7 , 31 , 27 , 5 , 2 , 17 , 25 ]
17 print(sort(L))
18 # affiche : [2, 5, 7, 17, 23, 25, 27, 31, 41]

```

Exercice 158. Écrire un algorithme en Python permettant de trier une liste selon l'algorithme du tri à bulles.

Solution.

```

1 # coding : utf-8
2 def bubble_sort(L) :
3
4     for i in range(len(L)) :
5         for j in range(len(L) - 1, i, -1) :
6             if L[j] < L[j - 1] :
7                 L[j], L[j - 1] = L[j - 1], L[j]
8
9     return L
10
11 # Exemple
12 L = [ '17' , '13' , '22' , '43' , '5' , '11' , '19' ]
13 print(bubble_sort(L))
14 # affiche : ['11', '13', '17', '19', '22', '43', '5']

```

Exercice 159. Écrire un algorithme en Python permettant de trier une liste selon l'algorithme du tri par sélection.

Solution.

```

1 def sort(L) :
2
3     for i in range(len(L) - 1) :
4         minimum = i
5
6         # Compare i and i + 1 element
7         for j in range(i + 1, len(L)) :
8             if L[j] < L[minimum] :
9                 minimum = j
10
11         if minimum != i :
12             L[i], L[minimum] = L[minimum], L[i]
13
14     return L
15
16 # Exemple
17 L = [23, 11, 9 , 47, 31, 49 , 7, 22 ]
18 print(sort(L))
19 # la sortie est : [7, 9, 11, 22, 23, 31, 47, 49]

```

Exercice 160. Écrire un algorithme Python permettant d'inverser l'ordre des éléments d'une liste en utilisant la méthode **reverse()**. Exemple si $L = [\text{'Java'}, \text{'Python'}, \text{'PHP'}, \text{'C++}]$, l'algorithme renvoie la liste : $[\text{'C++'}, \text{'PHP'}, \text{'Python'}, \text{'Java'}]$

Solution.

```

1 #coding : utf-8
2 def reverseList(L) :
3
4     # inverser la la liste L
5     L.reverse()
6
7     return L
8
9 # Exemple
10 L = [ 'Java' , 'Python' , 'PHP' , 'C++' ]
11 print(reverseList(L)) # la sortie est : [ 'C++', 'PHP', '
    Python', 'Java' ]

```

Exercice 161. Reprendre l'exercice précédent sans utiliser la méthode `reverse()` en python.

Solution.

```

1 #coding : utf-8
2 def reverseList(L) :
3
4     # initialisation de la liste inversée
5     lReverse = []
6     # parcourir les éléments de la liste L et les insérer
7     # d'une façon inversée
8     for item in L :
9         lReverse.insert(0 , item )
10
11     return lReverse
12
13 # Exemple
14 L = [ 'Java' , 'Python' , 'PHP' , 'C++' ]
15 print(reverseList(L)) # la sortie est : [ 'C++', 'PHP', '
    Python', 'Java' ]

```

Exercice 162. Etant donné une liste d'entiers `L`, écrire un algorithme en Python qui permet de multiplier les éléments de `L` d'index pair par 2 et ceux d'index impair par 3. Exemple si `L = [3 , 2 , 7 , 11 , 5 , 3]`, l'algorithme renvoie la liste `[6 , 6 , 14 , 33 , 10 , 9]`.

Solution.

```

1 #coding : utf-8
2 def mult(L) :

```

```

3     # initialiser la liste recherchée
4     listMult = []
5     for n in L :
6         if n%2 == 0 :
7             listMult.append(2*n)
8         else :
9             listMult.append(3*n)
10    return listMult
11
12
13    # Exemple
14    L = [3 , 2 , 7 , 11 , 5 , 3]
15    print(mult(L)) # affiche [9, 4, 21, 33, 15, 9]

```

Exercice 163.

Écrire un programme en python permettant d'extraire d'une liste de chaînes, la liste des chaînes contenant au moins deux voyelles. Exemple si `L = ["Java" , "Python" , "Dart" , "MySQL"]`, le programme renvoie la liste `["Java" , "Python"]`.

Solution.

```

1  #coding: utf-8
2  def vowels(L) :
3      # initialisation de la liste des mots contenant au
4      # moins deux voyelles
5      lVowels = []
6      # define vowels list
7      vowelsList = ['a' , 'e' , 'y' , 'u' , 'i' , 'o']
8
9      for u in L :
10         # initialiser le nombre de voyels contenu dans u
11         numberVowels = 0
12         for x in u :
13             if x in vowelsList :
14                 numberVowels = numberVowels + 1
15             if numberVowels >= 2 :
16                 lVowels.append(u)
17         return lVowels
18
19    # Exemple
20    L = ["Java" , "Python" , "Dart" , "MySQL" ]
21    print(vowels(L)) # la sortie est : ['Java', 'Python']

```

Exercice 164. Écrire un algorithme Python qui transforme la liste $L = [1, 2, 3, 4, 5]$ en la liste : $[1, [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5]]$

Solution.

1 Facile laissé au lecteur !

Exercice 165. Écrire un algorithme en Python permettant de remplacer les éléments d'une liste de chaînes de caractères en leurs longueurs. Exemple : si $L = ["Python", "Django", "Numpy", "Sympy"]$, l'algorithme renvoie la liste $[6, 6, 5, 5]$

Solution.

```

1 #coding : utf-8
2 def replaceByLength (L) :
3     # initialization de la liste recherchée
4
5     listLength = []
6
7     # parcourir les éléments de L et les remplacer par
    leurs longueurs
8     for x in L :
9         listLength.append(len(x))
10    return listLength
11
12 # Exemple
13 L = ["Python" , "Django" , "Numpy" , "Sympy"]
14 print (replaceByLength(L)) # la sortie est : [6, 6, 5, 5]
```

Exercice 166. Écrire un algorithme en Python permettant d'extraire la liste des nombres entiers d'une liste de nombres. Exemple : si $L = [2.5, 11.54, 3, 7.35, 5, 6.5, 9]$, l'algorithme renvoie la liste $[3, 5, 9]$

Solution.

```

1 #coding : utf-8
2 def extractInt (L) :
3     # initialization de la liste des entiers
4
5     listInt = []
6
```



```

7     # parcourir les éléments de L et rechercher les
    nombres entiers
8     for x in L :
9         if type(x) == int :
10             listInt.append(x)
11     return listInt
12
13 # Exemple
14 L = [2.5 , 11.54 , 3 , 7.35 , 5 , 6.5 , 9]
15 print (extractInt(L)) # la sortie est : [3, 5, 9]

```

Exercice 167. Écrire sans utiliser aucune fonction prédéfinie un algorithme en Python sous forme de fonction qui prends en paramètre une liste d'entiers L et qui renvoie True si la liste est rangée dans l'ordre croissant et False si non.

Solution.

```

1 #coding: utf-8
2 def testSort(L) :
3     # initialiser un compteur
4     count = 0
5     for i in range(0 , len(L)-1) :
6         if L[i] > L[i+1] :
7             count = count + 1
8     if count > 0 :
9         return False
10    else :
11        return True
12
13
14 # Exemple
15 L1 = [13 , 11 , 43 , 7 , 5 , 25 , 9]
16 L2 = [3 , 11 , 21 , 27 , 35 , 41 , 59]
17 print (testSort(L1)) # affiche False
18 print (testSort(L2)) # affiche True

```

Exercice 168. Écrire un algorithme Python qui détermine pour un entier donné, la liste des carrés parfaits compris entre 1 et n. Exemple si n = 100 l'algorithme renvoie la liste [1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 , 100]

Solution.

```

1 # coding : utf-8

```

```

2 # fonction qui test si un nombre est carré parfait ou non
3 def testCarreParfait (n) :
4     # initialization compteur count
5     count = 0
6
7     for i in range(1 , n+1) :
8         if i*i == n :
9             count = count + 1
10
11     if count > 0 :
12         return True
13     else :
14         return False
15
16 # fonction qui détermine la liste des carrés parfaits
17 def listCarresParfait(n) :
18     # initialisation de la liste des carrés parfaits
19     lparfait = []
20     for i in range(1, n+1) :
21         if testCarreParfait(i) :
22             lparfait.append(i)
23     return lparfait
24
25 #Exemple
26 n = 100
27 print (listCarresParfait(n))
28 # affiche : [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

```

Exercice 169. Écrire un algorithme Python qui détermine pour un entier n donné la liste des couples d'entiers (p, q) vérifiant : $p^2 + q^2 = n$

Solution.

```

1 #coding : utf-8
2 def listTuple(n) :
3     # initialisation de la liste des couples (p,q)
4     tupleList = []
5
6     # parcourir l'ensemble des couples (p,q) tels que p <=
7     n et q <= n
8     for p in range(0 , n+1) :
9         for q in range(0 , n+1) :
10             if p**2 + q**2 == n :
11                 tupleList.append((p,q))
12     return tupleList
13
14 # Exemple n = 100
15 print (listTuple(1000)) # affiche [(10, 30), (18, 26), (26,
16                             18), (30, 10)]

```

Exercice 170. Écrire un algorithme Python permettant de tester si une liste est symétrique en utilisant la méthode `reverse()`. Exemple : pour $L1 = [2, 5, 11, 5, 2]$ l'algorithme renvoie True et pour $L2 = [2, 23, 11, 51, 7]$ l'algorithme renvoie False.

Solution.

```

1 #coding : utf-8
2 def symetric(L) :
3     # créer une copie de L
4     reverseL = L.copy()
5
6     # création de l'inverse de la liste
7     reverseL.reverse()
8     if reverseL == L :
9         return True
10    else :
11        return False
12
13 #Exemple
14 L1 = [2 , 5 , 11 , 5 , 2]
15 L2 = [2 , 23 , 11 , 51 , 7]
16
17 print(symetric(L1)) # affiche True
18 print(symetric(L2)) # affiche False

```

Exercice 171. Écrire un algorithme Python qui transforme une liste d'entiers $L = [n_1, n_2, n_3, \dots, n_p]$ en ajoutant 1 au premier élément, 2 au deuxième élément, 3 au troisième élément, ... , p au $p^{\text{ème}}$ élément. L'algorithme doit renvoyer à la fin la liste $[n_1 + 1, n_2 + 2, n_3 + 3, \dots, n_p + p]$.

Solution.

```

1 #coding : utf-8
2 def translatedList(L) :
3     # initialisation de la liste tradatée
4     TList = []
5
6     for i in range(0, len(L) ) :
7         TList.append(L[i] + i+1)
8     return TList
9
10 # Exemple
11 L = [ 3 , 1 , 7 , 11 , 2]
12 print(translatedList(L)) # affiche : [4, 3, 10, 15, 7]

```

Exercice 172. Écrire une fonction qui prend en paramètre un élément x et une liste L , et renvoie la liste des positions de x dans L . La fonction doit renvoyer la liste vide si l'élément x ne figure pas dans la liste L .

Solution.

```

1 #coding : utf-8
2 def positions(x , L) :
3     # initialisation de la liste des positions de x dans L
4     LPositions = []
5
6     for i in range(0 , len(L)) :
7         if L[i] == x :
8             LPositions.append(i)
9     return LPositions
10 #Exemple
11 x = 3
12 y = 5
13 L = [2 , 3 , 7 , 3 , 11 , 9 , 3 , 4]
14 print(positions(x , L)) # affiche [1, 3, 6]
15 print(positions(y , L)) # affiche []

```

Exercice 173. Écrire une fonction qui prend en paramètre une chaîne de caractères s et une autre chaîne T , et renvoie la liste des positions d'un motif s dans la chaîne T . La fonction doit renvoyer la liste vide si le motif s ne figure pas dans la chaîne de caractères T .

Solution.

```

1 # coding : utf-8
2 #fonction qui détermine la liste des positions de s dans
   le texte T
3 def listPositions(T , s) :
4     # initialisation de la liste des occurrence de s dans
       T
5     lPosition = []
6     for i in range(0 , len(T)) :
7         # on test si l'occurrence se trouve à la position
           i
8         if T[i : len(s)+i] == s :
9             lPosition.append(i)
10    return lPosition
11 # Exemple
12 T = "Python is the most popular language and the most used
       language"
13 s = 'most'
14 print(listPositions(T , s)) # affiche : [14, 44]

```

Exercice 174. Écrire un algorithme Python qui demande à l'utilisateur de saisir successivement 7 nombres entiers et de lui afficher la liste des nombres saisis tout en ignorant les nombres répétés. Exemple si l'utilisateur saisi les nombres : 2 , 11 , 3 , 2 , 3 , 5 , 2. l'algorithme renvoie la liste [2 , 11 , 3 , 5].

Solution.

```

1 #coding : utf-8
2 #Initialisation de la liste des nombres entrés
3 listNumbers = []
4 for i in range(0 , 7) :
5     n = int(input("Enter a number"))
6     if n not in listNumbers :
7         listNumbers.append(n)
8     print("La liste des nombres entrés au clavier est : " ,
          listNumbers)
```

Exercice 175. Écrire un algorithme Python sous forme de fonction qui prends en paramètre une liste L formée de nombres entiers et qui renvoie la liste des index des entiers pairs figurant dans la liste L. Exemple pour L = [3 , 4 , 7 , 12 , 43 , 22 , 9] l'algorithme renvoie la liste [1 , 3 , 5].

Solution.

```

1 #coding : utf-8
2 def listIndex(L) :
3     # initialisation de la liste des index des entiers
    pairs
4     evenIndex = []
5     for i in range(0 , len(L)) :
6         # on test si l'élément L[i] est pair avant de l'
        ajouter à la liste
7         if L[i]%2 == 0 :
8             evenIndex.append(i)
9     return evenIndex
10
11 # Exemple
12 L = [3 , 4 , 7 , 12 , 43 , 22 , 9]
13 print("la liste des index des entiers pairs est : " ,
      listIndex(L))
```

Exercice 176. Écrire un algorithme en Python qui détermine l'index du maximum dans une liste donnée en utilisant la méthode max(). Exemple

si $L = [22, 7, 11, 41, 14, 9]$, l'algorithme renvoie l'index du maximum qui 3.

Solution.

```

1 #coding : utf-8
2 def indexMax(L) :
3     m = max(L)
4     index_max = L.index(m)
5     return index_max
6
7 # Exemple
8 L = [22 , 7 , 11 , 41 , 14 , 9 , 11]
9 print("L'index du maximum est : " , indexMax(L))
10 # La sortie du programme est : L'index du maximum est : 3

```

Exercice 177. Reprendre l'exercice précédent (Exercice 176) sans utiliser la méthode `max()`.

Solution.

```

1 #coding : utf-8
2 def maxList(L) :
3     # initialisation de l'index du maximum de la liste
4     i = 0
5     for j in range(0 , len(L)) :
6         if L[i] < L[j] :
7             # on remplace i par j
8             i = j
9     return i
10
11 # Exemple
12 L = [3 , 4 , 7 , 12 , 43 , 22 , 9]
13 print("l'index du maximum est : " , maxList(L))
14 # la sortie du programme est : l'index du maximum est : 4

```

Exercice 178. Étant donné une liste formée des moyennes des élèves `liste_moy`, écrire un algorithme qui détermine le premier index de la note au dessous de la moyenne. Exemple si `liste_moy = [12, 17, 10, 7, 11, 14, 15, 9]`, l'algorithme renvoie l'index 3.

Solution.

```

1 # coding : utf-8
2 def moyInf10(L) :
3     # initialisation de l'index à rechercher
4     index_moy = 0
5     for i in range(0 , len(L)) :
6         if L[i] < 10 :
7             index_moy = i
8             break
9     return index_moy
10
11 # Exemple
12 liste_moy = [12 , 17 , 10 , 7 , 11 , 14 , 15 , 9]
13 print (moyInf10(liste_moy))
14 #La sortie du programme est : 3

```

Exercice 179. Écrire en utilisant la méthode `count()`, un algorithme python sous forme de fonction qui prends en paramètre une liste d'entiers `L` et qui renvoie sans répétitions la liste des tuples **(`n` , `occ_n`)** formée des éléments `n` de `L` et de leurs occurrence `occ_n`. Exemple : si `L = [22 , 7 , 14 , 22 , 7 , 14 , 7 , 14 , 11 , 7]`, l'algorithme renvoie la liste `[(22 , 2) , (7 , 4) , (14 , 3) , (22 , 1) , (11 , 1)]`

Solution.

```

1 # coding : utf-8
2 def tuples_items_Occ(L) :
3     # initialisation de la liste des tuples
4     listItemsOcc = []
5     for x in L :
6         if (x , L.count(x)) not in listItemsOcc :
7             listItemsOcc.append((x , L.count(x)))
8     return listItemsOcc
9
10 # Exemple
11 L = [22 , 7 , 14 , 22 , 7 , 14 , 7 , 14 , 11 , 7]
12 print (tuples_items_Occ(L))
13 #La sortie du programme est :
14 #[(22 , 2) , (7 , 4) , (14 , 3) , (11 , 1)]

```

Exercice 180. Reprendre l'exercice précédent sans utiliser la méthode `count()` ni aucune autre méthode prédéfinie en Python.

Solution.

```
1 # Exercice laissé aucteur !
```

Exercice 181. Écrire un programme en Python qui transforme une chaîne de caractères en une liste sans utiliser la méthode `split()` ni aucune autre méthode prédéfinie en Python.

Solution.

```
1 # Exercice laissé aucteur !
```

Exercice 182. Écrire un programme en Python qui détermine la liste des mots contenant deux caractères identiques successifs dans une chaîne de caractère `s`. Exemple si `s = "Python is the most recommended programming language"`, l'algorithme renvoie la liste `["recommended", "programming"]`.

Solution.

```
1 # coding : utf-8
2 # fonction qui teste si un mot contient deux caractères
  succèsifs
3 def succèsifs(word) :
4     # initialisation d'un compteur
5     counter = 0
6     for i in range(0 , len(word) - 1) :
7         if word[i] == word[i+1] :
8             counter = counter + 1
9     if counter == 0 :
10        return False
11    else :
12        return True
13 # fonction qui détermine la liste des mots contenant au
  moins 2 caractères succèsifs
14 def listSuccèsifs(T) :
15     # convertir la variable chaîne de caractères T en une
  liste
16     L = T.split()
17     # initialisation de la liste des mots contenant au
  moins 2 caractères succèsifs
18     list_succèsifs = []
19
20     for word in L :
21         if succèsifs(word) :
```



```

22         list_successifs.append(word)
23     return list_successifs
24
25 # Exemple
26 word = 'Python programming language is used by a large
        community'
27 print(listSuccessifs(word))
28 # La sortie du programme est : ['programming', 'community',

```

Exercice 183. Écrire un algorithme en python qui remplace les éléments d'indice pairs par 2 et les éléments d'indices impairs par 1.

Solution.

```

1 # coding: utf-8
2 # fonction qui remplace les éléments d'indice pairs par 2
  et les éléments d'indices impairs par 1
3 def replace(L) :
4
5     # initialisation de la liste recherchée
6     list_0_1 = []
7
8     for i in range(0, len(L)) :
9         if i%2 == 0 :
10             list_0_1.append(2)
11         else :
12             list_0_1.append(1)
13     return list_0_1
14
15 # Exemple
16 L = ["Python" , "Java" , "C++" , "C#" , "Javascript"]
17 print(replace(L)) # affiche : [2, 1, 2, 1, 2]

```

Exercice 184. Étant donné un entier n, écrire un algorithme en python qui renvoie la liste des entiers inférieurs ou égale à n composées de deux chiffres et dont le chiffre des dizaine est divisible par 3.

Solution.

```

1 # coding: utf-8
2 def listDiv3(n) :
3     # initialisation de la liste
4     lDiv3 = []
5     for i in range(0 , n+1) :

```

```

6         # on teste si le chiffre des dizaines q = i//10
        est divisible par 3
7         if (i//10) % 3 == 0 and (i//10) != 0 :
8             lDiv3.append(i)
9         return lDiv3
10
11 # Exemple
12 n = 50
13 print(listDiv3(n))
14 # La sortie est : [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]

```

Exercice 185. Écrire un programme en Python qui permet de déplacer les valeurs nulles d'une liste à la fin de la liste tout en gardant l'ordre des autres éléments non nuls. Exemple si la liste est

$L = [7, 0, 11, 5, 0, 21, 0, 2, 0, 0, 9]$, l'algorithme renvoie la liste : $[7, 11, 5, 21, 2, 9, 0, 0, 0, 0, 0]$.

Solution.

```

1 # coding : utf-8
2 L = [7 , 0 , 11 , 5 , 0 , 21 , 0 , 2 , 0 , 0 , 9]
3 L_search = []
4 # Extraire la liste des éléments non nuls de L et la liste
  des éléments nuls
5 L_nuls = []
6 L_non_nuls = []
7
8 for x in L :
9     if x == 0 :
10         L_nuls.append(x)
11     else :
12         L_non_nuls.append(x)
13
14 # on additionne les deux listes
15 L_search = L_non_nuls + L_nuls
16
17 print(L_search)

```

Exercice 186. Étant donné une liste L, écrire un algorithme en python qui renvoie la liste des éléments entiers de L. Exemple : si $L = ["Python3", 91, "Java2", 95]$, l'algorithme renvoie la liste : $[91, 95]$.

Solution.

```

1 # coding : utf-8
2 def listInteger(L) :
3     # initialisation de la liste des entiers
4     lInteger= []
5     for x in L :
6         # on teste si l'élément x est un entier
7         if type(x) == int :
8             lInteger.append(x)
9     return lInteger
10
11 # Exemple
12 L = ["Python3" , 91 , "Java2" , 95]
13 print(listInteger(L))
14 # La sortie est : [91, 95]

```

Exercice 187. Etant donné une list L, écrire un algorithme en python permettant de convertir une liste en une chaîne de caractère sans utiliser aucune méthode prédéfinie autre que la méthode str(). Exemple si L = ["Python", "created on", 91, "by Guido Van Rosam"], l'algorithme renvoie la chaîne de caractères s = "Python created on 91 by Guido Van Rosam".

Solution.

```

1 # coding : utf-8
2 L = ["Python" , "created on", 91 , "by Guido Van Rosam"]
3
4 # initialisation de la chaîne s
5 s = ""
6 for word in L :
7     s = s + str(word) + " "
8
9 print (s) # output : 'Python created on 91 by Guido Van
           Rosam'

```

Exercice 188. (*) Étant donné une liste L, écrire un algorithme en python qui renvoie la liste des chiffres contenus au sein des éléments de la liste L. Exemple si L = ["Python3", 91, "Java2", 95], l'algorithme renvoie la liste [3, 9, 1, 2, 9, 5].

Solution.

```

1 #coding : utf-8
2 L = ["Python3" , 91 , "Java2" , 95]
3 # initialisation de la liste des chiffres
4 list_digit = []
5
6 #parcourir les éléments de la liste L
7 for word in L:
8     word = str(word)
9     # parcourir les éléments de la chaîne word et
      rechercher les éléments entiers
10    for x in word:
11        if x.isdigit():
12            x = int(x)
13            list_digit.append(x)
14
15 print(list_digit)
16 # La sortie est : [3 , 9 , 1 , 2 , 9 , 5]

```

Exercice 189. (*) Étant donné une liste L , écrire un algorithme en python qui renvoie la liste des chiffres contenus au sein des éléments de la liste L sans répétition. Exemple si $L = ["Python3" , 91 , "Java2" , 95]$, l'algorithme renvoie la liste $[3 , 9 , 1 , 2 , 5]$

Solution.

```

1 # coding : utf-8
2 L = ["Python3" , 91 , "Java2" , 95]
3
4 # convertir L en une chaîne de caractères
5 s = " ".join(str(x) for x in L)
6
7 # initialisation de la liste des chiffres sans répétition
8 list_digit = []
9
10 for x in s:
11     if x.isdigit() and x not in list_digit:
12         list_digit.append(x)
13
14 print("La liste des chiffres est :", list_digit)
15 # la sortie du programme est : La liste des chiffres est :
    ['3', '9', '1', '2', '5']

```

Exercice 190. (*) Étant donné une liste d'entiers $L = [n_1, n_2, n_3, \dots, n_p]$, écrire un algorithme en python qui renvoie la liste : $L_{sum} = [n_1, n_1 + n_2, n_1 + n_2 + n_3, \dots, n_1 + n_2 + n_3 + \dots + n_p]$.

Solution.

```

1 # coding : utf-8
2 # fonction qui calcule la somme des éléments d'une liste
3 def sumList(L) :
4     # initialisation de la somme des élément de la liste
5     s = 0
6     for x in L :
7         s = s + x
8     return s
9
10 # fonction qui determine la liste L_sum
11 def sum_list(L) :
12     # initialisation de la liste L_sum
13     L_sum = []
14     for i in range(0 , len(L)) :
15         L_sum.append(sumList(L[0:i]))
16
17     return L_sum
18
19 # Exemple
20 L = [3 , 2 , 5 , 1 , 0 , 7]
21 print("L_sum = " , sum_list(L))

```

Exercice 191. Écrire un algorithme en python sous forme de fonction qui transforme une chaîne de caractères texte T en une liste où les mots contenant des chiffres sont placés à la fin de la liste. Exemple si **T = "Python_1 created in 1991. Currently it is in version Python_3.9"**, l'algorithme renvoie la liste : ['created', 'in', 'Currently', 'it', 'is', 'in', 'version', 'Python_1', '1991.', 'Python_3.9'].

Solution.

```

1 # coding : utf-8
2 # fonction qui teste si un mot contient un chiffre ou non
3 def digit_in_word(word) :
4     # initialisation d'un compteur
5     counter = 0
6     for x in word :
7         if x.isdigit() :
8             counter = counter + 1
9
10     if counter > 0 :
11         return True
12     else :
13         return False
14

```

```

15 # fonction qui transforme une chaîne en une liste en
    décalant les mots contenant des chiffres à la fin
16 def moveDigit(T) :
17     # transformation de la chaîne texte T en une liste
18     L = T.split()
19
20     # initialisation de la liste des mots qui ne
    contiennent aucun chiffre
21     L_without_digit = []
22
23     # initialisation de la liste des mots qui contiennent
    au moins un chiffre
24     L_with_digit = []
25
26     # parcourir les éléments de la liste L et rechercher
    les mots qui contiennent des chiffres
27     for word in L :
28         if digit_in_word(word) :
29             L_with_digit.append(word)
30         else :
31             L_without_digit.append(word)
32
33     result = L_without_digit + L_with_digit
34
35     return result
36
37 # Exemple
38 T = "Python_1 created in 1991. Currently it is in version
    Python_3.9"
39 print(moveDigit(T))
40 # la sorite est :
41 #['created', 'in', 'Currently', 'it', 'is', 'in', 'version',
    'Python_1', '1991.', 'Python_3.9']

```

Exercice 192. (**) Écrire un algorithme en python qui transforme une liste d'entiers $L = [n_1, n_2, n_3, \dots, n_p]$ en une liste dont les éléments sont les moyennes partielles :

$[n_1, \text{moyenne}(n_1, n_2), \text{moyenne}(n_1, n_2, n_3), \dots, \text{moyenne}(n_1, n_2, \dots, n_p)]$.

Solution.

```

1 # coding: utf-8
2 # Fonction qui calcul la moyenne d'une liste de nombres
3 def average_list(L) :
4     n = len(L)
5     # initialisation de la somme des éléments de la liste
6     sum = 0
7     for x in L :

```

```

8         sum = sum + x
9         return sum/n
10
11 # Fonction qui détermine la liste des moyennes des sommes
    partielles
12 def average_sum(L) :
13     # initialisation de la list des sommes partielles
14     l_average = []
15     for i in range(1 , len(L)) :
16         l_average.append(average_list(L[0:i]))
17
18     return l_average
19
20 # Exemple
21 L = [10 , 14 , 12 , 10 , 13 , 15]
22 print(average_sum(L))
23 # La sortie est : [10.0, 12.0, 12.0, 11.5, 11.8]

```

Exercice 193. (*) Écrire un algorithme en python qui transforme une liste d'entiers $L = [n_1, n_2, n_3, \dots, n_p]$ en la liste des factorielles : $[n_1!, n_2!, n_3!, \dots, n_p!]$.

Solution.

```

1 # fonction qui calcul factorielle d'un entier
2 def facto(n) :
3     f = 1
4     for i in range(1 , n + 1) :
5         f = f*i
6     return f
7
8 # fonction qui transforme la liste L = [n1 , n2 , n3
    , ..., np] en [n1! , n2! , n3! , ..., np!]
9 def list_facto( L) :
10     # initialisation de la liste [n1! , n2! , n3! , ..., np
    !]
11     l_facto = []
12
13     for n in L :
14         l_facto.append( facto(n) )
15
16     return l_facto
17
18 # Exemple
19 L = [5 , 2 , 1 , 3 , 4]
20 print("Liste des factorielle = " , list_facto(L))
21 # La sortie du programme est : Liste des factorielle =
    [120, 2, 1, 6, 24]

```

Exercice 194. (*) Écrire un algorithme en python qui permet d'extraire d'une chaîne de caractère la liste des mots dont le premier caractère est identique au dernier. Exemple si `s = "radar numéro 212"`, l'algorithme renvoie la liste `['radar', '212']`.

Solution.

```

1 # coding : utf-8
2 # fonction qui teste si le premier caractère est identique
  au dernier
3 def first_end( s ) :
4     if s[0] == s[-1] :
5         return True
6     else :
7         return False
8
9 # Fonction qui détermine la list des mots dont le premier
  caractère est identique au dernier
10 def list_first_end(s) :
11     # convertir la chaîne s en une liste :
12     L = s.split()
13     # initialisation de la liste recherchée
14     l_first_end = []
15
16     for word in L :
17         if first_end(word) :
18             l_first_end.append(word)
19     return l_first_end
20
21 # Exemple
22 s = "radar numéro 212"
23 print("La list recherchée est : " , list_first_end(s))
24 # La sortie est : La list recherchée est : ['radar',
  '212']

```

Exercice 195. Etant donnée une liste formée de chaînes de caractères. Écrire un algorithme en python qui transforme les mots de cette liste en des listes. Exemple pour `L = ["Python", "Java", "Javascript"]`, l'algorithme renvoie la liste : `[['P', 'y', 't', 'h', 'o', 'n'], ['J', 'a', 'v', 'a'], ['J', 'a', 'v', 'a', 's', 'c', 'r', 'i', 'p', 't']]`.

Solution.

```

1 L = ["Python", "Java", "Javascript"]
2 # initialisation de la liste qu'on cherche
3 L_searched = []
4 for word in L :

```



```

5     w = list(word)
6     L_searched.append(w)
7
8     print(L_searched)

```

Exercice 196. Écrire un programme Python qui permet d'extraire d'une variable chaîne de caractères la liste formée des mots contenant uniquement des majuscules. Exemple si `s = "Le langage PYTHON est beaucoup utilisé en IOT"`, l'algorithme renvoie la liste `L = ["PYTHON", "IOT"]`.

Solution.

```

1  # coding : utf-8
2  # fonction qui determine si une chaîne de caractères est
   complètement en majuscules
3  def allMaj(s) :
4      # initialiser un compteur
5      counter = 0
6      for x in s :
7          if not x.isupper() :
8              counter = counter + 1
9      if counter > 0 :
10         return False
11     else :
12         return True
13 # fonction qui renvoie la liste des mots tout en
   majuscules
14 def wordMaj(s) :
15     # initialiser la liste des mots en majuscules
16     listMaj = []
17     # convertir la chaîne s en une liste
18     L_s = s.split()
19
20     for word in L_s :
21         if allMaj(word) :
22             listMaj.append(word)
23     return listMaj
24
25 # Exemple
26 s = "Le langage PYTHON est beaucoup utilisé en IOT"
27 print(wordMaj(s)) # affiche : ['PYTHON', 'IOT']

```

Exercice 197. Écrire un programme en Python qui prend en entrée une chaîne de caractère `s` et qui renvoie la liste des mots sans majuscule contenus dans `s`. Exemple si `s = "Le langage Python a été crée par Guido"`

Van Rossum" , l'algorithme renvoie la liste : ['langage', 'a', 'été', 'créé', 'par'].

Solution.

```

1 # coding : utf-8
2
3 # fonction qui determine si une chaîne de caractères
  contient une majuscule
4 def Maj(s) :
5     # initialiser un compteur
6     counter = 0
7     for x in s :
8         if x.isupper() :
9             counter = counter + 1
10
11     if counter > 0 :
12         return True
13     else :
14         return False
15
16 # fonction qui renvoie la liste des mots sans majuscules
17 def withoutMaj(s) :
18     # initialiser la liste cherchée
19     list_without_maj = []
20
21     # convertir le texte en une liste :
22     L = s.split()
23
24     for word in L :
25         if Maj(word) == False :
26             list_without_maj.append(word)
27
28     return list_without_maj
29
30 # Exemple
31 s = "Le langage Python a été créé par Guido Van Rossum"
32 print(withoutMaj(s)) # affiche : ['langage', 'a', 'été', '
  créé', 'par']

```

Exercice 198. Écrire un programme python qui transforme la liste : **L** = ["Python Programming" , "Java programming" , "C++ Programming"] en un fichier texte dont les lignes sont les éléments de la liste L.

Solution.

```

1 #coding :utf-8
2 L = ["Python Programming" , "Java programming" , "C++
  Programming"]

```

```

3 # ouvrir un fichier en mode write
4 f = open("myFile.txt" , 'w')
5
6 # ajouter les lignes au fichier myFile.txt
7 for line in L:
8     f.write( line + "\n")
9
10 f.close()

```

Exercice 199. On appelle palindrome, un mot dont le premier caractère est identique au dernier, exemple 'radar' , 'laval'...Écrire un programme python qui supprime tous les palindromes d'une liste de chaîne de caractères. Exemple pour la liste $L = ["voiture" , "radar" , "maison" , "laval"]$, l'algorithme renvoie la liste : $L = ["voiture" , "maison"]$

Solution.

```

1 #coding :utf-8
2
3 # fonction qui détermine si un mot est un palindrome
4 def palindrome(s) :
5     if s[0] == s[-1] :
6         return True
7     else :
8         return False
9
10 # algorithme qui supprime tous les palindromes d'une liste
    de chaînes
11 L = ["voiture" , "radar" , "maison" , "laval"]
12 # initialisation de la liste sans palindrome
13 sans_palindrome = []
14
15 for mot in L:
16     if palindrome(mot) == False :
17         sans_palindrome.append(mot)
18 print("Liste sans palindrome : " , sans_palindrome)
19 # affiche : Liste sans palindrome : ['voiture' , 'maison']

```

Exercice 200. Écrire un algorithme Python permettant d'inverser une liste sans utiliser la méthode reverse().

Solution.

```

1 #coding : utf-8

```

```
2 def reverseList(L) :  
3     # initialisation de la liste inverse  
4     reverseL = []  
5     for i in range(0 , len(L)) :  
6         reverseL.insert(0 , L[i])  
7     return reverseL  
8 #Exemple  
9 L = [4 , 11 , 57 , 7]  
10 print(reverseList(L)) # affiche [7, 57, 11, 4]
```

3. Exercices sur les algorithmes designs python

Exercice 201. Écrire un algorithme en langage Python qui demande à l'utilisateur de saisir un entier n et de lui afficher un triangle d'étoiles ayant pour nombre de ligne l'entier n tapé par l'utilisateur comme l'indique le design suivant :

```
*  
**  
***  
****  
*****  
*****  
*****
```

Solution.

```
1 n = int(input("saisir le nombre de lignes : "))  
2  
3 symbol = '*'
```

```

4  for i in range(0 , n) :
5      print(symbol)
6      symbol = symbol + '*'

```

Exercice 202. Écrire un algorithme en langage Python qui demande à l'utilisateur de saisir un entier n et de lui afficher un triangle d'étoiles inversé ayant pour nombre de ligne l'entier n tapé par l'utilisateur comme l'indique le design suivant :

```

* * * * *
* * * *
* * *
* *
*

```

Solution.

```

1  n = int(input("saisir le nombre de lignes : "))
2
3  for j in range(n , 0 , -1) :
4      for i in range(j , 0 , -1) :
5          print('*', end = ' ')
6      print('\n')

```

Exercice 203. Écrire un algorithme en Python qui demande à l'utilisateur de saisir un entier n et de lui afficher à l'écran un triangle d'étoiles isocèle dont le nombre de ligne sera exactement l'entier n tapé par l'utilisateur comme l'indique le design suivant :


```

1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10
3 4 5 6 7 8 9 10
4 5 6 7 8 9 10
5 6 7 8 9 10
6 7 8 9 10
7 8 9 10
8 9 10
9 10
10

```

Solution.

```

1 number_lines = 10
2 for i in range(0 , number_lines + 1) :
3     for j in range(i + 1 , number_lines + 1) :
4         print( j , end = " ")
5     print(" ")

```

Exercice 205. Écrire un programme python qui affiche un triangle d'étoiles comme le montre la figure suivante :


```

*
**
***
****
*****
******
*******
*****
****
***
**
*

```

Solution.

```

1 number_lines = 20
2 s = ""
3 for i in range(1 , number_lines) :
4     if i <= 10 :
5         s = s + "*"
6         print(s)
7     else :
8         s = s[0 : len(s) - 1]
9         print(s)

```

4. Exercices d'arithmétiques en Python

Exercice 206. Écrire un programme en langage Python qui demande à l'utilisateur de saisir un entier n et de lui renvoyer le message indiquant que le nombre tapé est premier ou non selon l'entier saisi au clavier.

Solution.

```
1 # coding : utf-8
2 n = int(input("Saisir la valeur de n: "))
3 # initialisation du nombre de diviseurs
4 numberDivisors = 0
5
6 # parcourir les entiers de 2 à n-1
7 for i in range(2, n):
8     if n%i == 0:
9         numberDivisors = numberDivisors + 1
10 if numberDivisors > 0:
11     print(n, " n'est pas premier !")
12 else:
13     print(n, " est premier")
```

Exercice 207. Écrire un algorithme python sous forme de fonction qui prend en argument deux entiers m et n qui retourne la liste de tous les diviseurs communs à m et n .

Solution.

```
1 #coding : utf-8
2 def diviseursCommuns(m,n) :
```

```

3     # initialisation de la liste des diviseurs communs
4     listDivCommuns = []
5     # Parcourir les entiers 1 , 2, .... , n
6     for i in range(1, n+1):
7         if n%i == 0 and m%i == 0 :
8             listDivCommuns.append(i)
9     return listDivCommuns
10
11 # Exemple (m , n) = ( 54 , 72)
12 print("La liste des diviseurs communs de 54 et 72 est " ,
        diviseursCommuns(54,72))
13 # La sortie est : La liste des diviseurs communs de 54 et
        72 est  [1, 2, 3, 6, 9, 18]

```

Exercice 208. Écrire un algorithme Python sous forme de fonction qui prend en argument un entier n et renvoie le plus grand entier premier inférieur ou égal à n . Exemple pour $n = 15$, la fonction renvoie le plus grand nombre premier inférieur ou égal à 15, soit 13.

Solution.

```

1  # Fonction permettant de tester la primalité d'un nombre
2  def testPrim(n) :
3      # initialisation de la liste des diviseurs de n
4      listDiv = []
5      # parcourir les entiers de 1 à n et récupérer les
        diviseurs de n
6      for i in range(1,n+1):
7          if n%i == 0:
8              listDiv.append(i)
9      if len(listDiv) == 2:
10         return True
11     else:
12         return False
13
14 def greatestPrim(n) :
15     p = n
16     while ( not testPrim(p)) :
17         p = p - 1
18     return p
19
20 # Exemple
21 print("Le plus grand nombre premier inférieur ou égale à
        15 est : " , greatestPrim(15))
22 #La sortie est : Le plus grand nombre premier inférieur ou
        égale à 15 est : 13

```

Exercice 209. Écrire un algorithme Python sous forme de fonction qui prend un entier n comme argument et renvoie le plus petit entier premier supérieur ou égal à n . Exemple pour $n = 8$, la fonction renvoie le plus petit nombre premier supérieur ou égal à 8 qui est 11.

Solution.

```

1 #coding : utf-8
2 # Fonction permettant de tester la primalité d'un nombre
3 def testPrim(n) :
4     # initialisation de la liste des diviseurs de n
5     listDiv = []
6     # parcourir les entiers de 1 à n et récupérer les
7     # diviseurs de n
8     for i in range(1,n+1) :
9         if n%i == 0 :
10             listDiv.append(i)
11     if len(listDiv) == 2 :
12         return True
13     else :
14         return False
15
16 def smallPrim(n) :
17     p = n
18     while ( not testPrim(p)) :
19         p = p + 1
20     return p
21
22 # Exemple
23 print("Le plus petit nombre premier supérieur ou égale à 8
24     est : " , smallPrim(152))

```

Exercice 210. Écrire un algorithme en Python sous forme de fonction qui prend en argument un **entier positif** n et retourne la liste de tous les tuples (u, v) d'entiers tels que : $u^2 + v^2 \leq n$.

Solution.

```

1 def listNumbers(n) :
2     # initialisation de la liste recherché.
3     l = []
4     for u in range(0 , n) :
5         for v in range(0 , n) :
6             if u**2 + v**2 <= n :
7                 l.append((u,v))
8     return l
9
10 # Exemple pour n = 57
11 print(listNumbers(57))

```

Exercice 211. Écrire un programme en Python qui détermine la liste des diviseurs impairs du nombre **3570** qui sont **multiples** de **3** et contenus dans l'intervalle **[500, 2500]**.

Solution.

```

1 def listDivisors(n):
2     # intialisation de la liste des diviseurs de n.
3     l = []
4     for i in range(1, n):
5         if ( n%i == 0 and i%3 == 0 and i%2 == 1 and i >=
6             500 and i <= 2500):
7             l.append(i)
8     return l
9 # Exemple pour n = 3570
10 print(listDivisors(3570))
# la sortie est : [1785]
```

Exercice 212. Écrire un programme en Python qui détermine le plus petit diviseur strictement supérieur à 1 d'un entier positif donné n.

Solution.

```

1 def smallestDivisor(n):
2     # intialisation du plut petit diseur de n
3     d = 2
4     while ( n%d != 0):
5         d = d + 1
6     return d
7 # Exemple
8 print("Le plus petit diviseur de 15 est : ",
9       smallestDivisor(15))
```

Exercice 213. Écrire un programme en Python qui recherche le plus grand diviseur stricte d'un entier positif donné n. Exemple pour n = 18, le plus grand diviseur de n est 9.

Solution.

```

1 def biggestDivisor(n):
2     # intialisation du plut grand diseur de n
3     d = n-1
4     while ( n%d != 0):
5         d = d - 1
```

```

6         return d
7     # Exemple
8     print("Le plus grand diviseur de 18 est : " ,
          biggestDivisor(18))

```

Exercice 214. Écrire une fonction en python qui prends en arguments deux nombres entiers **a** et **b** et qui renvoie un couple d'entiers formé par :

- 1) - Le **quotient** **q** de la division euclidienne de **a** par **b** (sans utiliser l'opérateur `//`)
- 2) - Le **reste** **r** de la division euclidienne de **a** par **b** (sans utiliser l'opérateur `%`)

Solution.

```

1  # coding : utf-8
2  # Création de la fonction quotient() :
3  def quotient(a,b) :
4      q = 1
5      if a < b :
6          return 0
7      else :
8          while ( b*q < a ) :
9              q = q + 1
10         return (q-1 , a - b*(q-1))
11
12 # Test de la fonction quotient
13 print("Le quotient et le reste de la division euclidienne
    de 19 par 4 est (q , r) = " , quotient(19,4))
14 # Affiche : Le quotient et le reste de la division
    euclidienne de 19 par 4 est (q , r) = (4, 3)

```

Exercice 215. Écrire un algorithme en Python qui demande à l'utilisateur de taper deux nombres entiers non nuls **a** et **b** et lui renvoie :

- 1) - Le **PGCD** de **a** et **b** sans utiliser aucune fonction prédéfinie en python.
- 2) - Le **PPCM** de **a** et **b** sans utiliser aucune fonction prédéfinie en python.

Solution.

Algorithme de calcul du PGCD

```

1 #coding : utf-8
2 def plusGrangDivCom(a,b) :
3     # on prend d = a comme valeur initiale
4     d = a
5
6     # tant que d n'est pas un diviseur commun
7     while (a%d != 0 or b%d !=0) :
8         # on décrement le d
9         d = d - 1
10    return d
11
12 # on test la fonction
13 print('Le plus grand diviseur commun de 9 et 12 est d =
    ', plusGrangDivCom(9,12))
14 # La sortie est : Le plus grand diviseur commun de 9 et 12
    est d = 3

```

Algorithme de calcul du PPCM

```

1 #coding : utf-8
2 def plusPetitMultCom(a,b) :
3     # on prends m = a comme initiale valeur
4     m = a
5
6     # tant que m n'est pas un multiple commun
7     while (m%a != 0 or m%b !=0) :
8         # on incrémente le m
9         m = m + 1
10    return m
11
12 # on test la fonction
13 print('le plus petit multiple commun de 9 et 6 est m = ',
    plusPetitMultCom(9 , 6))
14 #La sortie est : le plus petit multiple commun de 9 et 6
    est m = 18

```

Exercice 216. Écrire un algorithme sous forme de fonction en Python qui prends en argument un **entier n** et qui renvoie **True** si l'entier n est **premier** et **False** si **non**.

Solution.

```

1 #coding : utf-8
2 def primNumber(n) :
3     # On initialise le nombre de diviseur de n à 0
4     numberOfDivisors = 0
5     for i in range(1,n+1) :
6

```

```

7         # tant que i est un diviseur de n on incrémente le
          nombre de diviseurs : numberOfDivisors
8         if n%i == 0 :
9             numberOfDivisors = numberOfDivisors + 1
10        # le nombre n est premier si et seulement si
          numberOfDivisors == 2
11        if numberOfDivisors == 2 :
12            return True
13        else :
14            return False
15    # Test de la fonction
16    print(primNumber(7)) # affiche : True
17    print(primNumber(6)) # affiche : False

```

Exercice 217. Écrire un algorithme en Python sous forme de fonction qui prends deux nombres m et n en paramètres ($m < n$) et qui renvoie une liste formée de tous les nombres premiers compris entre m et n . Exemple pour $m = 10$ et $n = 20$ la fonction doit renvoyer [11, 13, 17, 19]

Solution.

```

1  # fonction qui test si un nombre est premier ou non
2  def primNumber(n) :
3      # take initial value of number of divisors
4      numberOfDivisors = 0
5      for i in range(1,n+1) :
6
7          # tant que i est un diviseur de n, on incrémente
            numberOfDivisors
8          if n%i == 0 :
9              numberOfDivisors = numberOfDivisors + 1
10         # le nombre n est premier si numberOfDivisors == 2
11         if numberOfDivisors == 2 :
12             return True
13         else :
14             return False
15
16  # fonction qui détermine la liste des nombres premiers
          dans un intervalle [m,n]
17  def listPrimNumbers(m,n) :
18      listPrim = []
19      for i in range(m , n+1) :
20
21          # we test if i is prim or not
22          if (primNumber(i)) :
23              # we add i to the list if it is a prim number
24              listPrim.append(i)
25      return listPrim

```



```

26
27 # Testing the function
28 print(listPrimNumbers(10,20))
29 # The output is [11, 13, 17, 19]

```

Exercice 218. Écrire un algorithme sous forme de fonction en Python qui prends en argument deux nombres entiers a et b et lui renvoie **True** si les nombres sont **premiers entre eux** et **False** si **non**.

Solution.

```

1 # coding : utf-8
2 def prim(n,m) :
3     # compteur qui compte le nombre de diviseurs communs à
4     m et n
5     nombreDiv = 0
6     for i in range(1,n+1) :
7         # Si i est un diviseur commun à m et n on
8         incrémente le compteur nombrDiv
9         if ( n%i == 0 and m%i == 0) :
10             nombreDiv = nombreDiv + 1
11         # Si le nombre des diviseurs communs à m et n est
12         = 1
13         # alors m et n sont premiers entre eux
14         if (nombreDiv == 1) :
15             return True
16         else :
17             return False
18 # On teste l'algorithme
19 print(prim(5,7)) # affiche True car 5 et 7 sont premiers
20 entre eux
21 print(prim(8, 12)) # affiche False car 8 et 12 ne sont pas
22 premiers entre eux

```

Exercice 219. Écrire un algorithme en Python qui demande à l'utilisateur de taper deux nombres entiers a et b premiers entre eux et lui renvoie un tuple (u,v) vérifiant : $ua + vb = 1$ (identité de Bezout)

Solution.

```

1 # coding : utf-8
2 # Saisir les valeurs des entiers a et b
3 a = int(input("Saisir la valeur de l'entier a : "))
4 b = int(input("Saisir la valeur de l'entier b : "))

```

```

5 # Initialiser la valeur de v à 1
6 v = 1
7 # On cherche l'entier u tel que 1-vb soit un multiple de a
8 # Tant que 1-vb n'est pas un multiple de a, on incrémente
  l'entier v
9 while ( (v*b-1)%a != 0 ) :
10     v = v + 1
11 # Maintenant l'égalité ua+vb=1 se traduit par
12 u = int((1-v*b)/a)
13 print("Le couple qui vérifie ua + vb = 1 est : ",(u,v))

```

Exercice 220. Déterminer la liste des diviseurs impairs du nombre **3570** qui sont multiples de **3** et contenus dans l'intervalle **[500, 2500]**.

Solution.

```

1 # coding : utf-8
2 l = []
3 n = 3570
4 # on parcourt les entiers de 1 à n
5 for i in range (1, n+1):
6     # on sélectionne les entiers vérifiant les conditions
      de l'hypothèse
7     if ( n%i == 0 and i%3==0 and i<=2500 and i>=500) :
8         # on ajoute les entiers vérifiant l'hypothèse
9         l.append(i)
10 print("La liste est : ", l)
11 # Affiche : La liste est : [510, 714, 1785]

```

Exercice 221. Écrire un algorithme en python qui demande à l'utilisateur de saisir un entier n et qui renvoie la liste des entiers positifs p tels que $n < p^2 < 3 * n^2$.

Solution.

```

1 # coding : utf-8
2 n = int(input("Entrez la valeur de n"))
3
4 # initialiser la liste demandée
5 L = []
6 for p in range(n , 3*n**2) :
7     if p**2 < 3*n**2 :
8         L.append(p)
9 print("La liste demandée est : " , L)

```

Exercice 222. Écrire un algorithme python qui demande à l'utilisateur de saisir un **entier** n et de lui renvoyer la liste des entiers $p \leq n$ dont le dernier chiffre **3** ou **7**. Exemple : si l'utilisateur saisi l'entier $n = 20$, l'algorithme lui renvoie la liste : $L = [3, 7, 13, 17]$

Solution.

```

1 # coding : utf-8
2 # on demande à l'utilisateur de saisir un entier n
3 n = int(input("Entrer la valeur de n"))
4
5 # initialisation de la liste recherchée
6 L_3_7 = []
7 # parcourir les entiers 1 , 2 , .... , n
8 for i in range(1 , n + 1) :
9     # tester si le dernier chiffre de i est égale à 3 ou à
10     7
11     if i%10 == 3 or i%10 == 7 :
12         L_3_7.append(i)
13 # afficher la liste
14 print(L_3_7)

```

Exercice 223. Écrire un algorithme en python permettant d'afficher tous les nombres entiers positif composés de deux chiffres $n = pq$ telle que la somme des chiffres $p+q$ est un **multiple** de **5**.

Solution.

```

1 # coding : utf-8
2 # initialisation de la liste recherchée
3 L = []
4 for n in range(10 , 100) :
5
6     # chiffre des unités
7     q = n % 10
8
9     # chiffres des dizaines
10    p = int((n - q)//10)%10
11
12    if (p + q)%5 == 0 :
13        L.append(n)
14 print(L)
15 # affiche : [14, 19, 23, 28, 32, 37, 41, 46, 50, 55, 64,
16             69, 73, 78, 82, 87, 91, 96]

```

Exercice 224. Écrire un algorithme en python qui détermine la **liste** des **diviseurs premiers** d'un **nombre entier**.

Solution.

```

1 # coding : utf-8
2 # fonction qui test si un nombre est premier ou non
3 def testPrim(n) :
4     #nombre des diviseurs de n
5     number_div = 0
6     for i in range(1,n+1) :
7         if n%i == 0 :
8             number_div = number_div + 1
9     if number_div == 2 :
10         return True
11     else :
12         return False
13
14 # fonction qui determine la liste des diviseurs premiers
    de n
15 def listDivPrim(n) :
16     # initialisation de la liste des diviseurs premiers
17     l_divPrim = []
18     for i in range(2 , n+1) :
19         if testPrim(i) and n%i == 0 :
20             l_divPrim.append(i)
21     return l_divPrim
22
23 # Exemple
24 n = 42
25 print(listDivPrim(n)) # affiche [2, 3, 7]
```

Exercice 225. Écrire un algorithme en python qui demande à l'utilisateur de saisir un nombre entier $n \leq 100$ et de lui afficher la **liste** des nombres **premiers** $\leq n$ de la forme **pq** avec **p+q** est **pair**. Exemple si l'utilisateur saisi $n = 40$, le programme renvoie la liste : **[11, 13, 17, 19, 31, 37]**.

Solution.

```

1 # coding : utf-8
2 # fonction qui test si un nombre est premier ou non
3 def testPrim(n) :
4     #nombre des diviseurs de n
5     number_div = 0
6     for i in range(1,n+1) :
7         if n%i == 0 :
8             number_div = number_div + 1
9     if number_div == 2 :
```

```

10         return True
11     else :
12         return False
13
14 # liste des nombre premier dont la somme des chiffres est
    pair
15 # initialisation de la liste recherchée
16 listPrim = []
17 n = int(input("Entrer la valeur de n : "))
18
19 for i in range(10, n) :
20     # chiffre des unités de i
21     p = i%10
22     # chiffre des dizaines de i
23     q = (i-p) // 10
24     if testPrim(i) and (p + q)%2 == 0 :
25         listPrim.append(i)
26 print(listPrim) # affiche : [11, 13, 17, 19, 31, 37, 53,
    59, 71, 73, 79, 97]

```

Exercice 226. Écrire un programme en python sous forme de fonction qui prends en paramètres deux entiers **m** et **n** et qui renvoie la **liste des diviseurs communs** à **m** et **n**. Exemple si **n = 18** et **m = 27** , la fonction renvoie la liste : **[1, 3, 9]**

Solution.

```

1 # coding : utf-8
2 def listDivCommuns (n , m) :
3     # initialisation de la liste des diviseurs communs
4     divCommuns = []
5     for k in range(1 , n+1) :
6         if n%k == 0 and m%k == 0 :
7             divCommuns.append(k)
8     return divCommuns
9
10 # exemple
11 n , m = 18 , 27
12 print(listDivCommuns(n , m))
13 # affiche : [1, 3, 9]

```

Exercice 227. Écrire un programme python qui détermine la liste des **diviseurs premiers communs** à deux entiers.

Solution.

```

1 # fonction qui teste la primalité d'un nombre
2 def testPrim(n) :
3     #nombre des diviseurs de n
4     number_div = 0
5     for i in range(1,n+1) :
6         if n%i == 0 :
7             number_div = number_div + 1
8     if number_div == 2 :
9         return True
10    else :
11        return False
12
13 def listDivPrimCommuns (n , m) :
14     # initialisation de la liste des diviseurs communs
15     divPrimCommuns = []
16     for k in range(1 , n+1) :
17         if n%k == 0 and m%k == 0 and testPrim(k) :
18             divPrimCommuns.append(k)
19     return divPrimCommuns
20
21 # exemple
22 n , m = 42 , 70
23 print(listDivPrimCommuns(n , m))
24 # affiche : [2 , 7]

```

Exercice 228. Écrire un programme en python qui détermine pour un entier donné n , la liste des tuples (p, q) formés des entiers premiers p et q vérifiant $p + q = n$.

Solution.

```

1 # coding : utf-8
2
3 # fonction qui teste la primalité d'un nombre
4 def testPrim(n) :
5     #nombre des diviseurs de l'entier n
6     number_div = 0
7     for i in range(1,n+1) :
8         if n%i == 0 :
9             number_div = number_div + 1
10    if number_div == 2 :
11        return True
12    else :
13        return False
14
15 def listTuples ( n ) :
16     # initialisation de la liste des tuples
17     tuple_list = []

```

```

18     for p in range(1 , n+1) :
19         for q in range(1 , n+1) :
20             if testPrim(p) and testPrim(q) and p + q == n :
21                 tuple_list.append((p , q))
22     return tuple_list
23
24 # exemple
25 n = 18
26 print(listTuples(n ))
27 # affiche : [(5 , 13), (7 , 11), (11 , 7), (13 , 5)]

```

Exercice 229. Écrire un algorithme en python qui détermine la somme des diviseurs propres d'un entier n . Exemple pour $n = 10$, les diviseurs propres de n sont **1** , **2** et **5** et par conséquent l'algorithme retourne la somme **1 + 2 + 5 = 8**.

Solution.

```

1 # coding : utf-8
2
3 # fonction qui détermine la somme des diviseurs propre d'
  un entier
4 def sommeDiviseurs(n) :
5     # initialisation de la somme des diviseurs de n
6     somme = 0
7     for i in range(1, n) :
8         if n%i == 0 :
9             somme = somme + i
10    return somme
11
12 print("La sommedes diviseurs de 10 est : " , sommeDiviseurs
    (10))
13 # affiche : La sommedes diviseurs de 10 est : 8

```

Exercice 230. (nombres amis) Écrire un algorithme en python sous forme de fonction qui permet de tester si deux nombres m et n sont **amis** ou non.

Rappel : On dit que deux entiers positifs m et n sont **amis** ou **amicaux** si chacun des deux est égal exactement à la somme des diviseurs stricts (diviseurs autres que lui-même) de l'autre. Exemple $n = 220$, la somme des diviseurs de n (en éliminant 220) : $1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$ $m = 284$, la somme des diviseurs de m (en éliminant 284) : $1 + 2 + 4 + 71 + 142 = 220$ et par suite m et n sont amis.

Solution.

```

1 # coding : utf-8
2
3 # fonction qui détermine la somme des diviseurs propre d'
  un entier
4 def sommeDiviseurs(n) :
5     # initialisation de la somme des diviseurs de n
6     somme = 0
7     for i in range(1, n) :
8         if n%i == 0 :
9             somme = somme + i
10    return somme
11
12 # fonction qui teste si deux nombres sont amis ou non
13 def nombresAmis(m , n) :
14     if sommeDiviseurs(m ) == n and sommeDiviseurs(n) == m :
15         return True
16     else :
17         return False
18
19 # Exemple
20 m , n = 220 , 284
21 print(nombresAmis(m , n)) # affiche : True
22
23 p , q = 151 , 250
24 print(nombresAmis(p , q)) # affiche : False

```

Exercice 231. La suite de **Fibonacci** est une **suite récurrente** définie par :

$$\begin{cases} F_0 = 0 & F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n \end{cases}$$

Créer une fonction en python qui renvoie le $(n+1)^{\text{ème}}$ terme de la suite de **Fibonacci** pour un entier n donné, puis un algorithme qui affiche les **10 premiers** termes de la suite de **Fibonacci**.

Solution.

```

1 # coding : utf-8
2
3 # fonction qui renvoie le n ème terme de la suite de
  Fibonacci
4 def fibo (n) :
5     F0 , F1 = 0 , 1
6     if n <= 1 :
7         return n
8     else :

```



```

9         return fibo(n-1) + fibo(n-2)
10
11 # affichage des 10 premiers terme de la suite de fibonacci
12
13 for i in range(0 , 11):
14     print("F" , i , " = " , fibo(i))
15 # affiche :
16 """
17 F 0  =  0
18 F 1  =  1
19 F 2  =  1
20 F 3  =  2
21 F 4  =  3
22 F 5  =  5
23 F 6  =  8
24 F 7  =  13
25 F 8  =  21
26 F 9  =  34
27 F 10 =  55
28 """

```

Exercice 232. Écrire un programme en python sous forme de **fonction** qui prends en argument un **entier n** et qui retourne la **liste des tuples (p , q)** formés des entiers **p** et **q** vérifiant :

p et **q** sont deux **diviseurs** de **n** avec **p + q** est **premier**. Exemple si **n = 10** , l'algorithme renvoie la liste : [(1, 1), (1, 2), (2, 1), (2, 5), (5, 2)]

Solution.

```

1 # coding : utf-8
2
3 # fonction qui teste la primalité d'un nombre
4 def testPrim(n) :
5     #nombre des diviseurs de l'entier n
6     number_div = 0
7     for i in range(1,n+1):
8         if n%i == 0 :
9             number_div = number_div + 1
10    if number_div == 2 :
11        return True
12    else :
13        return False
14
15 # fonction qui determine la liste des couples (p,q)
16 def list_div_couple(n) :
17     # initialisation de la liste des diviseurs
18     list_div = []

```

```

19
20     for p in range(1 , n) :
21         for q in range(1 , n) :
22             # on teste la primalité de p + q et la
divisibilité de n par p et q
23             if testPrim(p + q) and n%p == 0 and n%q == 0 :
24                 list_div.append((p,q))
25     return list_div
26
27 # exemple
28 n = 18
29 print(list_div_couple(n))
30 # affiche : [(1, 1), (1, 2), (1, 6), (2, 1), (2, 3), (2, 9)
, (3, 2), (6, 1), (9, 2)]

```

Exercice 233. Écrire un programme en python qui détermine pour un entier n donné les couples (p, q) vérifiant : p et q sont premiers entre eux et $p < q \leq n$

Solution.

```

1 # coding : utf-8
2 # fonction qui renvoie le pgcd de deux nombres
3 def pgcd(a,b) :
4     # calcul du plus grand commun diviseur
5     if b==0:
6         return a
7     else :
8         r=a%b
9         return pgcd(b,r)
10
11 # liste des couples (p,q) p <= q <= n tels que p premier
avec q
12
13 def primTuple(n) :
14     # initialisation de la liste recherchée
15     listTuple = []
16     for p in range(1, n+1) :
17         for q in range(1, n+1) :
18             if pgcd(p , q) == 1 and p < q :
19                 listTuple.append((p,q))
20     return listTuple
21
22 # Exemple
23 n = 10
24 print(primTuple(n))
25 # affiche :
26 """

```

```

27 [(1, 2), (1, 3), (1, 4), (1, 5), (1, 6),
28 (1, 7), (1, 8), (1, 9), (1, 10), (2, 3),
29 (2, 5), (2, 7), (2, 9), (3, 4), (3, 5),
30 (3, 7), (3, 8), (3, 10), (4, 5), (4, 7),
31 (4, 9), (5, 6), (5, 7), (5, 8), (5, 9),
32 (6, 7), (7, 8), (7, 9), (7, 10), (8, 9),
33 (9, 10)]
34 """

```

Exercice 234. Écrire un programme en python qui renvoie pour un **entier n** donné les **couples (p, q)** vérifiant : **p et q sont positifs non nuls** et $p < q \leq n$ avec p et q ne sont pas premiers entre eux et $\text{pgcd}(p, q) \leq 10$

Solution.

```

1  # coding : utf-8
2
3  # fonction qui calcul le pgcd de deux nombres
4  def pgcd(a,b) :
5      # calcul du plus grand commun diviseur
6      if b==0 :
7          return a
8      else :
9          r=a%b
10         return pgcd(b,r)
11
12 # liste des couples (p,q) vérifiant pgcd(p,q) <=10
13 def lessThan10(n) :
14     # initialisation de la liste recherchée
15     listTuple = []
16     for p in range(1, n+1) :
17         for q in range(1, n+1) :
18             if pgcd(p , q) > 1 and pgcd(p , q) <= 10 and p
19                 < q :
20                 listTuple.append((p,q))
21     return listTuple
22
23 # Exemple
24 n = 10
25 print(lessThan10(n))
26 # affiche :
27 """
28 [(2, 4), (2, 6), (2, 8), (2, 10), (3, 6),
29 (3, 9), (4, 6), (4, 8), (4, 10), (5, 10),
30 (6, 8), (6, 9), (6, 10), (8, 10)]
31 """

```

Exercice 235. Écrire un programme en python qui renvoie pour un **entier** n donné les **tuples** (p, q) avec $p \leq n$ et $q \leq n$ dont le plus grand diviseurs commun $\text{pgcd}(p, q)$ est **premier**.

Solution.

```

1  # coding : utf-8
2
3  # fonction qui calcul le pgcd de deux nombres
4  def pgcd(a,b) :
5      # calcul du plus grand commun diviseur
6      if b==0:
7          return a
8      else :
9          # algorithme d'euclide recursif
10         r=a%b
11         return pgcd(b,r)
12
13 # fonction qui teste la primalité d'un nombre
14 def testPrim(n) :
15     # On initialise le nombre de diviseur de n à 0
16     numberOfDivisors = 0
17     for i in range(1,n+1) :
18
19         # tant que i est un diviseur de n on incrémente le
20         # nombre de diviseurs : numberOfDivisors
21         if n%i == 0 :
22             numberOfDivisors = numberOfDivisors + 1
23     # le nombre n est premier si et seulement si
24     numberOfDivisors == 2
25     if numberOfDivisors == 2 :
26         return True
27     else :
28         return False
29
30 # liste des couples (p,q) vérifiant pgcd(p,q) est premier
31 def tuple_pgcd_premier(n) :
32     # initialisation de la liste recherchée
33     listTuple = []
34     for p in range(1, n+1):
35         for q in range(1, n+1):
36             if testPrim(pgcd(p, q)) :
37                 listTuple.append((p,q))
38     return listTuple
39
40 # Exemple
41 n = 10
42 print(tuple_pgcd_premier(n))
43 # affiche :
44 """
45 [(2, 2), (2, 4), (2, 6), (2, 8), (2, 10), (3, 3),
46  (3, 6), (3, 9), (4, 2), (4, 6), (4, 10), (5, 5),

```

```

45 (5, 10), (6, 2), (6, 3), (6, 4), (6, 8), (6, 9),
46 (6, 10), (7, 7), (8, 2), (8, 6), (8, 10), (9, 3),
47 (9, 6), (10, 2), (10, 4), (10, 5), (10, 6),
48 (10, 8)]
49 """

```

Exercice 236. Écrire un algorithme en python sous forme de fonction qui prends en paramètres un entier naturel n et qui renvoie la liste des nombres premiers dont le chiffre des dizaines est premier. Exemple pour $n = 100$, l'algorithme renvoie la liste : **[23, 29, 31, 37, 53, 59, 71, 73, 79]**.

Solution.

```

1 # coding : utf-8
2
3 # fonction qui teste la primalité d'un nombre
4 def testPrim(n) :
5     # On initialise le nombre de diviseur de n à 0
6     numberOfDivisors = 0
7     for i in range(1,n+1) :
8
9         # tant que i est un diviseur de n on incrémente le
          nombre de diviseurs : numberOfDivisors
10        if n%i == 0 :
11            numberOfDivisors = numberOfDivisors + 1
12    # le nombre n est premier si et seulement si
    numberOfDivisors == 2
13    if numberOfDivisors == 2 :
14        return True
15    else :
16        return False
17
18    """fonction qui détermine la liste des nombres premiers
19    dont le chiffre des dizaines est premier
20    """
21    def listPrim(n) :
22        # initialisation de la liste des nombres premiers
23        listPrimNumbers = []
24
25        for p in range(1 , n) :
26            # chiffre des dizaines de p
27            p_dizaines = p//10
28            if testPrim(p) and testPrim(p_dizaines) :
29                listPrimNumbers.append(p)
30        return listPrimNumbers
31
32 # Exemple

```

```

33 n = 100
34 print(listPrim(n))
35 # affiche : [23, 29, 31, 37, 53, 59, 71, 73, 79]

```

Exercice 237. Écrire un programme en python qui détermine pour un entier $n \geq 10$ si le chiffre des **dizaines** de n est **premier** avec celui des **unités** de n .

Solution.

```

1 # fonction qui teste si deux nombres sont premiers entre
  eux ou non
2 def premiersEntreEux(m , n) :
3     # initialisation de la liste des diviseurs communs de
      m et de n
4     listCommunsDiv = []
5     for i in range(1 , m + 1) :
6         if n%i == 0 and m%i == 0 :
7             listCommunsDiv.append(i)
8     if len(listCommunsDiv) == 1 :
9         return True
10    else :
11        return False
12
13 # fonction qui teste si le chiffre des dizaines est
      premier avec celui des unités
14 def testNumbers (n) :
15     # chiffre des unités de n
16     n_unite = n%10
17     # chiffre des dizaines de n
18     n_diz = ((n - n_unite)//10)%10
19
20     # teste si le chiffre des dizaines est premier avec
      celui des unités
21     if premiersEntreEux(n_unite , n_diz) :
22         return True
23     else :
24         return False
25
26 # Exemple
27 n = 157
28 print(testNumbers(n)) # affiche : True
29
30 m = 239
31 print(testNumbers(m)) # affiche : False

```

Exercice 238. Écrire un algorithme en python qui détermine pour un entier $n \geq 10$ donné la liste des nombres entiers premiers $\leq n$ dont le chiffre des dizaines n'est pas premier !

Solution.

```

1  # fonction qui test si un nombre est premier ou non
2  def testPrim(n) :
3      #nombre des diviseurs de n
4      number_div = 0
5      for i in range(1,n+1) :
6          if n%i == 0 :
7              number_div = number_div + 1
8      if number_div == 2 :
9          return True
10     else :
11         return False
12
13 # fonction qui renvoie la liste des entiers premiers dont
    le chiffre des dizaines n'est pas premier
14 def listPrim (n) :
15     # intialisation de la liste demandée
16     listPrimNumber = []
17     for k in range(10 , n+1) :
18         # chiffre des unités de k
19         k_unite = k%10
20         # chiffre des dizaines de k
21         k_diz = ((k - k_unite)//10)%10
22
23         # on teste si k est premier et k_diz ne l'est pas
24         if testPrim(k) == True and testPrim(k_diz) ==
        False :
25             listPrimNumber.append(k)
26     return listPrimNumber
27
28 # Exemple
29 n = 88
30 print(listPrim(n))
31 # affiche : [11, 13, 17, 19, 41, 43, 47, 61, 67, 83]
```

Exercice 239. On dit qu'un nombre **entier naturel** est **distinct**, s'il est formé de chiffres tous distincts deux à deux. Exemple : $n = 123456$ est distinct. Tandis que $m = 224544$ ne l'est pas puisque les chiffres qui le composent ne sont pas tous distincts. Écrire un algorithme python sous forme de fonction qui prends en paramètre un entier n et renvoie **True** si l'entier n est distinct et **False** si non.

Solution.

```

1 # fonction qui teste si un entier est distinct
2 def distinct(n):
3     # initialisation de la valeur de retour
4     value = True
5     # convertir n en une chaine de caractères
6     s = str(n)
7     for x in s:
8         if s.count(x) > 1:
9             value = False
10            break
11     if value == True:
12         return True
13     else:
14         return False
15
16 #Exemple
17 n , m = 123456 , 2244566
18 print(distinct(n)) # affiche : True
19 print(distinct(m)) # affiche : False

```

Exercice 240. Écrire un algorithme en Python qui détermine la liste de tous les entiers composés de **deux chiffres** **pq** vérifiant : la somme **p+q** est un **diviseur** du produit **pq**.

Solution.

```

1 # initialisation de la liste demandée
2 list_numbers = []
3
4 for p in range(1 , 10):
5     for q in range(0 , 10):
6         if (p * q) % (p + q) == 0:
7             n = q + 10*p
8             list_numbers.append(n)
9
10 print(list_numbers)
11 # affiche : [10, 20, 22, 30, 36, 40, 44, 50, 60, 63, 66,
              70, 80, 88, 90]

```

Exercice 241. Écrire un programme en Python qui détermine la liste de tous les entiers composés de **deux chiffres** **pq** vérifiant : la somme **p+q** est un **diviseur premier** du produit **pq**.

Solution.


```

1 # fonction qui test si un nombre est premier ou non
2 def testPrimality(n) :
3     #nombre des diviseurs de n
4     number_div = 0
5     for i in range(1,n+1) :
6         if n%i == 0 :
7             number_div = number_div + 1
8     if number_div == 2 :
9         return True
10    else :
11        return False
12
13 # initialisation de la liste demandée
14 list_numbers = []
15
16 for p in range(1 , 10) :
17     for q in range(0 , 10) :
18         if (p * q) % (p + q) == 0 and testPrimality(p + q)
19             == True :
20             n = q + 10*p
21             list_numbers.append(n)
22
23 print(list_numbers)
24 # affiche : [20, 30, 50, 70]

```

Exercice 242. Écrire un programme en python qui détermine pour un entier donné n la somme de ses diviseurs premiers. Exemple pour $n = 10$, l'algorithme renvoie la somme $2 + 5 = 7$.

Solution.

```

1 # fonction qui test si un nombre est premier ou non
2 def testPrimality(n) :
3     #nombre des diviseurs de n
4     number_div = 0
5     for i in range(1,n+1) :
6         if n%i == 0 :
7             number_div = number_div + 1
8     if number_div == 2 :
9         return True
10    else :
11        return False
12
13 # fonction qui détermine la somme des diviseurs premiers d
14   'un entier
15 def sumDivPrim(n) :
16     # initialisation de la somme demandée
17     s = 0

```

```
17     for i in range(1 , n + 1) :
18         if testPrimality(i) == True and n%i == 0 :
19             s = s + i
20     return s
21
22 # Exemple
23 n = 10
24 print("La somme des diviseurs premiers = " , sumDivPrim(n)
25 )
26 # affiche : La somme des diviseurs premiers = 7
```

5. Exercices sur les dictionnaires Python

Exercice 243. Etant donnée un dictionnaire python dont les clés sont les noms des élèves et les valeurs sont les listes des notes

```
1 d = {"Aladin": [12, 15 , 17] , "Nathalie" : [15, 13 , 16]
      , "Robert" : [13, 15 , 11] }
```

Écrire un programme qui remplace les listes des notes par leurs moyennes.

Solution.

```
1 # coding : utf-8
2 d = {"Aladin": [12, 15 , 17] , "Nathalie" : [15, 13 , 16]
      , "Robert" : [13, 15 , 11] }
3
4 # initialisation du dictionnaire recherché
5 d_moyennes = dict({})
6
7 for key , value in d.items() :
8     # calcul de la moyenne
9     m = 0
10    for x in value :
11        m = m + x
12    m = m/len(value)
13
14    # ajouter la moyenne arrondie au dictionnaire
15    d_moyennes[key] = round(m , 2)
16
17 print(d_moyennes)
18 # affiche {'Aladin': 14.67, 'Nathalie': 14.67, 'Robert':
    13.0}
```

Exercice 244. Étant donnée un **dictionnaire** **d** dont les valeurs des **clés** sont des **listes** :

```
1 d = { 'a1': [21, 17, 22, 3], 'a2': [11, 15, 8, 13], 'a3':
      [7, 13, 2, 11] , 'a4': [22, 14, 7, 9] }
```

Écrire un programme Python qui permet de transformer le dictionnaire **d** en triant les listes.

Solution.

```
1 d = { 'a1': [21, 17, 22, 3], 'a2': [11, 15, 8, 13],
2       'a3': [7, 13, 2, 11] , 'a4': [22, 14, 7, 9] }
3 d_sorted = dict({})
4 for key , value in d.items() :
5     value.sort()
6     d_sorted[key] = value
7
8 print(d_sorted)
9 #affiche: { 'a1': [3, 17, 21, 22], 'a2': [8, 11, 13, 15], '
      a3': [2, 7, 11, 13], 'a4': [7, 9, 14, 22] }
```

Exercice 245. Écrire un programme en Python qui demande à l'utilisateur de saisir un texte et de lui renvoyer un dictionnaire dont les clés sont les mots du texte saisi et les valeurs sont les inverses des mots qui composent le texte. Exemple pour le texte **T = " Python est un langage facile "**, le programme doit renvoyer le dictionnaire :

```
1 d = { 'Python': 'nohtyp' , 'est': 'tse' , 'langage': '
      egagnal' , 'facile': 'elicaf' }
```

Solution.

```
1 # coding: utf-8
2 # demander à l'utilisateur de saisir un texte
3 T = input("Saisir un texte : ")
4
5 # Convertir le texte en une liste
6 liste_words = T.split()
7
8 # création du dictionnaire demandé
9 d = dict({})
10 for word in liste_words :
11     d[word] = word[::-1]
12
13 print("Le dictionnaire demandé est " , d)
14 # ce qui affiche pour le texte cité ci-dessus :
```

```
15 # Le dictionnaire demandé est {'Python': 'nohtyP', 'est':
    'tse', 'un': 'nu', 'facile': 'elicafe'}
```

Exercice 246. Écrire un programme en Python qui demande à l'utilisateur de saisir un texte et de lui renvoyer un dictionnaire dont les clés sont les mots du texte saisi et les valeurs sont les longueurs des mots qui composent le texte. Exemple pour le texte **T = 'Python est un langage de programmation'**, le programme doit renvoyer le dictionnaire :

```
1 d = {'Python': 6, 'est': 3, 'un': 2, 'langage': 7, 'de':
    2, 'programmation': 13}
```

Solution.

```
1 # coding: utf-8
2 # demander à l'utilisateur de saisir un texte
3 T = input("Saisir un texte : ")
4
5 # Convertir le texte en une liste
6 liste_words = T.split()
7
8 # création du dictionnaire demandé
9 d = dict({})
10 for word in liste_words:
11     d[word] = len(word)
12
13 print("Le dictionnaire demandé est " , d)
14 # ce qui affiche pour le texte cité ci-dessus:
15 # Le dictionnaire demandé est {'Python': 6, 'est': 3, 'un':
    2, 'langage': 7, 'de': 2, 'programmation': 13}
```

Exercice 247. Écrire un programme en Python qui demande à l'utilisateur de saisir un entier **n** et de lui renvoyer un dictionnaire dont les clés sont les entiers **1, 2, 3, n** et dont les valeurs sont les sommes **1, 1+2, 1+2+3, ..., 1+2+3+...+n**.

Solution.

```
1 # coding: utf-8
2 # demander à l'utilisateur de saisir un entier n
3 n = int(input("Tapez la valeur de n : "))
4 # création d'une fonction qui calcul la somme 1+2+...+n
5 def somme(n):
```

```

6     s = 0
7     for i in range(1 , n+1) :
8         s = s + i
9     return s
10
11 # création du dictionnaire demandé
12 d = dict({})
13 for i in range(1 , n+1) :
14     d[i] = somme(i)
15
16 print("Le dictionnaire demandé est " , d)
17 # quand l'utilisateur fait entrer 5, le programme affiche :
18 # Le dictionnaire demandé est {1: 1, 2: 3, 3: 6, 4: 10, 5:
    15}

```

Exercice 248. Écrire un programme en Python qui demande à l'utilisateur de saisir un entier n et de lui renvoyer un dictionnaire dont les clés sont les entiers $1, 2, 3, \dots, n$ et dont les valeurs sont $1!, 2!, 3!, \dots, n!$

Solution.

```

1 # coding: utf-8
2 # demander à l'utilisateur de saisir un entier n
3 n = int(input("Tapez la valeur de n : "))
4 # création d'une fonction qui calcul n!
5 def facto(n) :
6     fact = 1
7     for i in range(1 , n+1) :
8         fact = fact*i
9     return fact
10
11 # création du dictionnaire demandé
12 d = dict({})
13 for i in range(1 , n+1) :
14     d[i] = facto(i)
15
16 print("Le dictionnaire demandé est\n" , d)

```

Exercice 249. Écrire un programme en Python qui demande à l'utilisateur de saisir cinq nombres entiers de son choix et de lui renvoyer un dictionnaire dont les clés sont les entiers saisis et dont les valeurs sont les listes des diviseurs des nombres saisis. Exemple si l'utilisateur saisi les nombres : **14 , 7 , 8 , 6 , 13**, le programme renvoie le dictionnaire :

```
1 d = {14 : [1, 2, 7, 14] , 7 : [1,7] , 8 : [1, 2, 4, 8] , 6 :
      [1,3,6] , 13 : [1,13] }
```

Solution.

```
1 # coding : utf-8
2 # fonction qui détermine la liste des diviseurs d'un
  entier
3 def listDiv(n) :
4     # initialisation de la liste des diviseur de n
5     l = []
6     # parcourt des entiers 1 , 2 , 3 ,... , n
7     for i in range(1, n+1):
8         # si i est un diviseur de n on l'ajoute à la liste
9         if n%i == 0 :
10             l.append(i)
11     return l
12
13 # récupération des nombres tapés dans une liste python
14 typed_number = []
15 for i in range(0, 5) :
16     n = int(input("Tapez un nombre entier"))
17     typed_number.append(n)
18
19 # création du dictionnaire
20 d = dict({})
21 for n in typed_number :
22     d[n] = listDiv(n)
23
24 print(d)
```

Exercice 250. Écrire une fonction en Python qui prends en paramètre une liste de nombres entiers et qui renvoie un dictionnaire dont les clés sont les entiers de la liste et dont les valeurs sont 'pair' ou 'impair' selon la parité du nombre.

Solution.

```
1 # coding : utf-8
2 def listToDict(l) :
3
4     # création d'un dictionnaire vide pour récupérer les
      résultats
5     dictParity = dict()
6
7     # parcourir les éléments de la liste et tester leur
      parité
```

```

8     for x in l :
9         if x%2 == 0 :
10             dictParity[x] = 'Pair'
11         else :
12             dictParity[x] = 'Impair'
13     return dictParity
14 # Faire un test :
15 l = [24 , 14 , 3 , 36 , 41 , 22 , 15]
16 print(listToDict(l))
17 # La sortie est : {24: 'Pair', 14: 'Pair', 3: 'Impair', 36:
    'Pair', 41: 'Impair', 22: 'Pair', 15: 'Impair'}

```

Exercice 251. (*) Écrire un programme en Python qui demande à l'utilisateur de saisir 5 nombres entiers de son choix et de lui renvoyer un dictionnaire dont les clés sont les entiers saisis et dont les valeurs sont 'premier' ou 'non premier' selon l'entier saisi.

Solution.

```

1 # coding : utf-8
2 # création d'une fonction qui teste la primalité d'un
   nombre entier donné
3 def testPrim(n) :
4     # initialisation du nombre de diviseurs de n
5     numberDivisors = 0
6     # parcourir les entiers 1 2 3 ... n et sélectionner
       les diviseurs de n
7     for i in range(1 , n + 1) :
8         if ( n%i == 0 ) :
9             numberDivisors = numberDivisors + 1
10    if (numberDivisors == 2 ) :
11        return True
12    else :
13        return False
14
15 # création d'un dictionnaire qui va contenir les valeurs
   saisies au clavier
16 d = dict({})
17 for i in range(0 , 5) :
18     n = int(input("Type an integer : " ))
19     if (testPrim(n)) :
20         d[n] = "prime"
21     else :
22         d[n] = "not prime"
23 print(d)
24
25 # Teste de l'algorithme
26 # pour les valeurs saisies : 3, 5, 4, 11, 18, la sortie
   est :

```



```
27 # {3: 'prime', 5: 'prime', 4: 'not prime', 11: 'prime',
    18: 'not prime'}
```

Exercice 252. Écrire un programme en Python qui demande à l'utilisateur de saisir une chaîne de caractère, et de lui renvoyer un dictionnaire dont les clés sont les caractères de la chaîne saisie et les valeurs sont les positions des caractères dans la chaîne de caractères. Exemple pour la chaîne `s = "langage"`, le programme renvoie le dictionnaire :

```
1 d = {'l':0 , 'a':1 , 'n':2 , 'g':3 , 'e':6}
```

Solution.

```
1 #coding: utf-8
2 s = "langage"
3
4 # initialisation du dictionnaire recherché
5 d = dict({})
6
7 # parcourir les caractères de la chaîne s
8 for x in s:
9     d[x] = s.index(x)
10
11 print(d)
```

Exercice 253. Écrire un programme en Python qui demande à l'utilisateur de saisir une chaîne de caractère, et de lui renvoyer un dictionnaire dont les clés sont les caractères de la chaîne saisie et les valeurs sont les nombres d'occurrences des caractères dans la chaîne de caractères. Exemple pour la chaîne `s = "langage"`, le programme renvoie le dictionnaire :

```
1 d = {'l':1 , 'a':2 , 'n':1 , 'g':2 , 'e':1}
```

Solution.

```
1 #coding: utf-8
2 s = "langage"
3
4 # initialisation du dictionnaire recherché
5 d = dict({})
6
```

```

7 # parcourir les caractères de la chaîne s
8 for x in s :
9     d[x] = s.count(x)
10
11 print(d)

```

Exercice 254. Écrire un programme Python qui permet de créer à partir d'un entier entier n saisi au clavier , un dictionnaire formé des entiers de 1 à n et de leurs carrés. Exemple pour $n = 7$ le dictionnaire sera de la forme :

```
1 d = {1 : 1, 2 : 4, 3 : 9, 4 : 16, 5 : 25 , 6 : 36 , 7 : 49}
```

Solution.

```

1 # coding : utf-8
2 # on demande à l'utilisateur de saisir un entier n
3 n = int(input("Entrer la valeur de n"))
4
5 # on crée un dictionnaire vide qui va contenir les nombre
  n et leurs carrée
6 d = dict({})
7
8 # on fait le parcourt des entiers de 1 à n
9 for i in range(1 , n+1):
10     d[i] = i*i
11
12 print(d)

```

Exercice 255. On considère les trois dictionnaires Python qui regroupe la totalité du matériels informatiques :

```

1 dicPC={"HP" : 11 , "Acer" : 7 , "Lenovo" : 17 , "Del" : 23}
2 dicPhone={"Sumsung" : 22 , "Iphone" : 9 , "Other" : 13 }
3 dicTablette = {"Sumsung" : 15 , "Other" : 13}

```

Écrire un programme Python qui regroupe en concaténant ces trois dictionnaires en un seul avec deux méthodes différentes.

Solution.

1^{ère} méthode

```

1 # coding : utf-8
2 dicPC={"HP" : 11 , "Acer" : 7 , "Lenovo" : 17 , "Del" : 23}
3 dicPhone={"Sumsung" : 22 , "Iphone" : 9 , "Other" : 13 }
4 dicTablette = {"Sumsung" : 15 , "Other" : 13}
5
6 # on crée un dictionnaire vide qui va contenir les autres
  dictionnaires
7 dicTotal = {}
8
9 # on ajoute les dictionnaire un par un via la méthode
  update
10 dicTotal.update(dicPC)
11 dicTotal.update(dicPhone)
12 dicTotal.update(dicTablette)
13 print(dicTotal)

```

2^{ème} méthode :

```

1 # coding : utf-8
2 dicPC={"HP" : 11 , "Acer" : 7 , "Lenovo" : 17 , "Del" : 23}
3 dicPhone={"Sumsung" : 22 , "Iphone" : 9 , "Other" : 13 }
4 dicTablette = {"Sumsung" : 15 , "Other" : 13}
5 # on crée un dictionnaire qui va contenir les autres
6 dicTotal = {}
7 # on ajoute les dictionnaires via la boucle for
8 for d in [dicPC , dicTablette , dicPhone] :
9     dicTotal.update(d)
10
11 print(dicTotal)

```

Exercice 256. Étant donnée un dictionnaire en python dont les pairs (clé : valeur) sont les noms des élèves avec leurs âges : **d = {"Robert" : 17 , "Catherine" : 21 , "Majid" : 23 , "Farid" : 15 }**. Écrire un programme en python qui transforme le dictionnaire d en la liste : **L = [('Robert', 17) , ('Catherine', 21) , ('Majid', 23) , ('Farid', 15)]**.

Solution.

```

1 d = {"Robert" : 17 , "Catherine" : 21 , "Majid" : 23 , "
    Farid" : 15 }
2
3 # initialisation de la liste demandée
4 L = []
5
6 # parcourir les éléments du dictionnaire
7 for key , value in d.items() :
8     L.append((key , value))

```

```

9 # affichage de la liste
10 print (L)
11 # output : [('Robert', 17), ('Catherine', 21), ('Majid',
23), ('Farid', 15)]

```

Exercice 257. Écrire un programme en **Python** qui transforme le texte : **T = "les versions Python 3.9 et Python 3.10 sont beaucoup plus utilisées que les versions Python 2.x"** en un **dictionnaire** dont les **clés** sont les **mots** du texte **T** et le **valeurs** associés sont les **nombre**s d'**occurrences** des **mots** dans le texte.

Solution.

```

1 T = "les versions Python 3.9 et Python 3.10 sont beaucoup
   plus utilisées que les versions Python 2.x"
2
3 # initialisation du dictionnaire demandé
4 d = dict({})
5
6 # convertir s en une liste
7 L = T.split()
8
9 for word in L:
10     d[word] = L.count(word)
11
12 # affichage du dictionnaire
13 print(d)
14 """
15 output : {'les': 2, 'versions': 2, 'Python': 3, '3.9': 1,
16 'et': 1, '3.10': 1, 'sont': 1, 'beaucoup': 1,
17 'plus': 1, 'utilisées': 1, 'que': 1, '2.x': 1}
18 """

```

Exercice 258. On considère les **deux dictionnaires** suivants dont les **clés** sont les **noms** des **modules** de **formations** proposées et dont les **valeurs** des **clés** sont les **prix** associés en **Euro**. **Formation1** = {"Python" : 350 , "Django" : 400 , "PHP" : 320 , "Java" : 450}, **Formation2** = {"Python" : 570 , "Django" : 350 , "PHP" : 300 , "Java" : 570}. Créer un programme en **Python** qui prends en entré les deux dictionnaires et renvoie un autre **dictionnaire** nommé **Formation** formé des mêmes clés et dont les **valeurs** associées sont les **minimum** des **valeurs** des **prix** de **Formation1** et **Formation2**.

Solution.

```
1 Formation1 = {"Python" : 350 , "Django" : 400 , "PHP" :  
    320 , "Java" : 450}  
2 Formation2 = {"Python" : 570 , "Django" : 350 , "PHP" :  
    300 , "Java" : 570}  
3  
4 # initialisation du dictionnaire demandé  
5 Formation = dict({})  
6  
7 # parcourir les valeur du dictionnaire Formation1  
8 for key , value in Formation1.items():  
9     if Formation1[key] >= Formation2[key]:  
10         Formation[key] = Formation2[key]  
11     else:  
12         Formation[key] = Formation1[key]  
13  
14 # affichage du dictionnaire  
15 print(Formation)  
16 #output : {'Python': 350, 'Django': 350, 'PHP': 300, 'Java'  
    ' : 450}
```

6. Exercices sur les ensembles Python

Exercice 259. Ecrire un programme en Python qui renvoie l'intersection des deux ensembles $A=\{'a', 'b', 'c', 'd'\}$ et $B=\{'c', 'e', 'd', 'h'\}$ sans utiliser la méthode `intersection`.

Solution.

```
1 A={'a' , 'b' , 'c' , 'd'}
2 B={'c' , 'e' , 'd' , 'h'}
3
4 # initialiser l'intersection de A et B
5 C = set({})
6
7 # parcourir les éléments de A et verifier leurs
  appartenance à B
8 for x in A:
9     if x in B:
10         C.add(x)
11 print("L'intersection de A et B est C = " , C)
12 # affiche : L'intersection de A et B est C = {'c', 'd'}
```

Exercice 260. Ecrire un programme en Python qui renvoie la réunion des deux ensembles $A=\{'a', 'b', 'c', 'd'\}$ et $B=\{'c', 'e', 'd', 'h'\}$ sans utiliser la méthode `union` (on pourra utiliser la boucle `for` pour faire le parcourt des éléments de B et les ajouter à l'ensemble A)

Solution.

```
1 A={'a' , 'b' , 'c' , 'd'}
2 B={'c' , 'e' , 'd' , 'h'}
3
4 # parcourir les éléments de B et les ajouter à A
```

```

5 for x in B:
6     A.add(x)
7
8 print("La réunion de A et B est : " , A)
9 # affiche : La réunion de A et B est : {'e', 'd', 'a', 'b', 'c', 'h'}

```

Exercice 261. Etant donné deux ensembles $A=\{'a', 'b', 'c', 'd'\}$ et $B=\{'c', 'e', 'd', 'h'\}$. Ecrire un programme en langage Python qui renvoie leur **différence symétrique** sans utiliser la méthode `symmetric_difference()`.

Solution.

```

1 A={'a', 'b', 'c', 'd'}
2 B={'c', 'e', 'd', 'h'}
3
4 # initialiser la différence symétrique
5 symmetric_diff = set({})
6
7 # différence A - B
8 for x in A:
9     if x not in B:
10         symmetric_diff.add(x)
11
12 # différence B - A ( les éléments de B qui n'appartiennent pas à A)
13 for x in B:
14     if x not in A:
15         symmetric_diff.add(x)
16
17 # Afficher la différence symétrique
18 print("différence symétrique = " , symmetric_diff)
19 # output : différence symétrique = {'e', 'h', 'b', 'a'}

```

Exercice 262. Etant donné un ensemble $A = \{'a', 'b', 'c', 'd'\}$. Ecrire un algorithme en Python qui permet d'ajouter un élément 'x' à A sans utiliser la méthode `add()`.

Solution.

```

1 A = { 'a', 'b', 'c', 'd' }
2
3 # on ajoute 'x' à A via la méthode union()
4 A2 = A.union({'x'})

```

```

5
6 # afficher l'ensemble A2
7 print(A2) # output : {'a', 'd', 'b', 'c', 'x'}

```

Exercice 263. Ecrire un programme en Python qui permet de **supprimer** un élément d'un ensemble **A** sans sans utiliser les méthodes **discard()** et **remove()**.

Solution.

```

1 # définir un ensemble A
2 A = { 'a', 'b', 'c', 'd' }
3 # supprimer l'élément 'd' sans utiliser ni la méthode
   remove() ni la méthode discard()
4
5 #initialiser un ensemble vide B
6 B = set({})
7
8 # parcourir les éléments de A
9 for x in A:
10     if x != 'd':
11         B.add(x)
12
13 print("A\{'d'\} = ", B)
14 # output : A\{'d'\} =  {'b', 'c', 'a'}

```

Exercice 264. Ecrire un programme en langage Python qui renvoie l'intersection et la réunion des trois ensembles suivants :

```

1 A = {11 , 21 , 5 , 7 , 43 , 32 , 13 , 9}
2 B = {2 , 19 , 11 , 33 , 7 , 25 , 5 , 4}
3 C = {45 ,27 , 11 , 5 , 7 , 22 , 14 , 1}

```

Solution.

```

1 A = {11 , 21 , 5 , 7 , 43 , 32 , 13 , 9}
2 B = {2 , 19 , 11 , 33 , 7 , 25 , 5 , 4}
3 C = {45 ,27 , 11 , 5 , 7 , 22 , 14 , 1}
4
5 # Intersection des trois ensembles
6 I = A.intersection(B.intersection(C))
7 print(I) # output : {11, 5, 7}
8
9 # réunion des trois ensembles
10 R = A.union(B.union(C))
11 print(R) # output : {1, 2, 4, 5, 7, 9, 11, 13, 14, 19, 21,
   22, 25, 27, 32, 33, 43, 45}

```


Exercice 265. Reprendre l'exercice précédent (**Exercice 264**) sans utiliser les méthodes `intersection()` et `union()`.

Solution.

```

1 A = {11 , 21 , 5 , 7 , 43 , 32 , 13 , 9}
2 B = {2 , 19 , 11 , 33 , 7 , 25 , 5 , 4}
3 C = {45 , 27 , 11 , 5 , 7 , 22 , 14 , 1}
4
5 # Initialisation de l'intersection des trois ensembles
6 I = set({})
7
8 for x in A:
9     if x in B and x in C:
10         I.add(x)
11 # afficher l'intersection des trois ensembles
12 print(I)
13
14 # Initialisation de la réunion des trois ensembles
15 R = set({}) # output: {11, 5, 7}
16
17 for x in A:
18     for y in B:
19         for z in C:
20             R.add(x)
21             R.add(y)
22             R.add(z)
23 # afficher la réunion des trois ensembles A, B et C
24 print(R)
25 # output: {32, 33, 1, 2, 4, 5, 7, 9, 11, 43, 45, 14, 13,
    19, 21, 22, 25, 27}

```

Exercice 266. Ecrire un programme en Python qui renvoie l'ensemble des entiers **carrés parfaits** inférieur ou égaux à 100.

Solution.

```

1 # fonction qui teste si un nombre est un carré parfait
2 def perfectSquare(n) :
3     test = False
4     for i in range(0 , n+1) :
5         if i**2 == n:
6             test = True
7     return test
8
9 # initialisation de l'ensemble demandé
10 A = set({})
11
12 # parcourir les entiers de 1 à 100

```

```

13 for i in range(0 , 101):
14     # tester si 'i' est un carré parfait
15     if perfectSquare(i):
16         A.add(i)
17
18 # afficher l'ensemble des carrés parfaits
19 print("A = ", A)
20 # output: A = {0, 1, 64, 4, 36, 100, 9, 16, 49, 81, 25}

```

Exercice 267. Créer un algorithme en Python qui détermine l'ensemble des **entiers impairs** inférieur ou égaux à 100 qui sont **multiple de 3**.

Solution.

```

1 # initialisation de l'ensemble demandé
2 A = set({})
3
4 # parcourir les entiers de 0 à 100
5 for i in range(0 , 101):
6     if i%3 == 0 and i%2 != 0:
7         A.add(i)
8
9 # afficher l'ensemble des entiers impairs et multiple de 3
10 print("A = " , A)
11 #output: A = {33, 3, 99, 69, 39, 9, 75, 45, 15, 81, 51,
    21, 87, 57, 27, 93, 63}

```

Exercice 268. Ecrire un programme en langage python qui détermine l'ensemble des **nombre premiers** de 1 à 100.

Solution.

```

1 #fonction qui teste la primalité d'un nombre
2 def testPrim(n):
3     # initialisation du nombre de diviseur de n
4     numberDiv = 0
5     for i in range(1,n+1):
6         # si i est un diviseur de n on incrémente le
        nombre de diviseurs de n
7         if n%i == 0:
8             numberDiv = numberDiv + 1
9     # le nombre n est premier si et seulement si numberDiv
    == 2
10     if numberDiv == 2:
11         return True

```

```

12     else :
13         return False
14
15 # initialisation de l'ensemble des nombres premiers de 1 à
16   100
17 A = set({})
18
19 # parcourir l'ensemble des entiers de 1 à 100
20 for i in range(1, 101):
21     if testPrim(i):
22         A.add(i)
23
24 # afficher l'ensemble des nombres premiers
25 print("A = ", A)
26 """
27 output :
28 A = {2, 3, 5, 7, 11, 13, 17, 19, 23,
29      29, 31, 37, 41, 43, 47, 53, 59,
30      61, 67, 71, 73, 79, 83, 89, 97}
31 """

```

Exercice 269. Ecrire un programme en langage python qui prend en entrée un **texte T** et qui renvoie l'**ensemble des mots** qui commencent par la lettre **'p'**. Exemple sit **T = "python is the most popular programming language"**, le programme renvoie l'ensemble : **{'programming', 'popular', 'python'}**.

Solution.

```

1 T = "python is the most popular programming language"
2
3 # convertir le texte T en une liste
4 list_words = T.split()
5
6 # initialiser l'ensemble des mots qui commencent par la
7   lettre 'p'
8 begin_p = set({})
9
10 # parcourir les mots de la liste list_words
11 for word in list_words:
12     if word[0] == 'p':
13         begin_p.add(word)
14
15 # afficher l'ensemble des mots qui commencent par la
16   lettre 'p'
17 print(begin_p) # output : {'programming', 'popular', '
18   python'}

```

Exercice 270. Ecrire un programme qui demande à l'utilisateur de saisir 5 lettres successivement et de lui renvoyer l'ensemble python formé des lettres saisies.

Solution.

```

1 # initialisation de l'ensemble demandé
2 A = set({})
3 for i in range(0 , 5) :
4     x = input("Enter an element : ")
5     A.add(x)
6 print("Ensemble des éléments tapés A = " , A)
7 """
8 output :
9 Enter an element : a
10 Enter an element : g
11 Enter an element : z
12 Enter an element : y
13 Enter an element : k
14 Ensemble des éléments tapés A = {'y', 'k', 'g', 'z', 'a'}
15 """

```

Exercice 271. Ecrire un algorithme en python sous forme de fonction qui prend en paramètre un **couple (A,B)** formé de deux **ensembles A et B** et qui renvoie **'True'** si **A** est un sous ensemble de **B** et **'False'** si non.

Solution.

```

1 # fonction qui teste si un ensnsemble A est un sous
   ensemble d'un ensemble B
2 def subset(A , B) :
3     if A.issubset(B) :
4         return True
5     else :
6         return False
7
8 # Exemple
9 A1 = { 'a' , 'b' , 'c' , 'd' }
10 A2 = { 'a' , 'b' , 's' , 'k' }
11 B = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' }
12
13 print(subset(A1 , B)) # output : True
14 print(subset(A2 , B)) # output : False

```

$[\{\}, \{b\}, \{a\}, \{c\}, \{b, a\}, \{b, c\}, \{a, c\}, \{b, a, c\}]$.

Exercice 272. Reprendre l'exercice précédent sans utiliser la méthode `issubset()` des ensembles python.

Solution.

```

1 # fonction qui teste si un esnsemble A est un sous
  ensemble d'un ensemble B
2 def subset(A , B) :
3     # initialiser un compteur
4     counter = 0
5
6     # parcourir les éléments x de A et vérifier si x n'est
      pas dans B
7     for x in A:
8         if x not in B:
9             counter = counter + 1
10
11     if counter == 0 :
12         return True
13     else :
14         return False
15
16 # Exemple
17 A1 = { 'a' , 'b' , 'c' , 'd' }
18 A2 = { 'a' , 'b' , 's' , 'k' }
19 B = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' }
20
21 print(subset(A1 , B)) # output : True
22 print(subset(A2 , B)) # output : False

```

Exercice 273. Ecrire un algorithme en python qui prends en entrée l'ensemble $A = \{ 'a' , 'b' , 'c' \}$ et renvoie la liste formée de **toutes les parties de A**. Le programme doit renvoyer la liste : $\{ \{ \} , \{ 'b' \} , \{ 'a' \} , \{ 'c' \} , \{ 'b' , 'a' \} , \{ 'b' , 'c' \} , \{ 'a' , 'c' \} , \{ 'b' , 'a' , 'c' \} \}$.

Solution.

```

1 A = { 'a' , 'b' , 'c' }
2 # initialiser la liste demandée
3 L = []
4
5 # ajouter l'ensemble vide {}
6 L.append({})
7
8 # ajouter les singletons
9 for x in A:
10     L.append({x})
11

```

```
12 # ajouter les paires
13 for x in A:
14     for y in A:
15         if {x, y} not in L:
16             L.append({x, y})
17
18 # ajouter l'ensemble {'a' , 'b' , 'c' }
19 L.append(A)
20
21 # afficher la liste
22 print(L)
23
24 #output : [{}, {'b'}, {'c'}, {'a'}, {'b', 'c'}, {'b', 'a'},
           {'c', 'a'}, {'b', 'c', 'a'}]
```

7. Exercices sur les fichiers Python

Exercice 274. (création et écriture et lecture d'un fichier texte)

1) - Écrire un programme Python qui permet de créer un fichier nommé **myFile.txt** et d'écrire le texte **T="This file is created with Python !"**.

2) - Écrire un programme Python qui permet de lire le fichier **myFile.txt** et d'afficher son contenu à l'écran

Solution.

Question1

```
1 #coding :utf-8
2
3 # création du fichier myFile.txt en mode write
4 f = open("myFile.txt" , 'w')
5
6 # écrire sur le fichier
7 f.write("This file is created with Python !")
8
9 f.close()
```

Question2

```
1 #coding :utf-8
2
3 # ouverture du fichier myFile.txt en mode lecture
4 f = open("myFile.txt" , 'r')
5
6 # extraire le contenu du fichier
7 content = f.read()
8
9 # afficher le contenu du fichier
10 print(content)
11
12 f.close()
```

Exercice 275. (création, écriture et lecture d'un fichier texte)

1) - Écrire un programme Python qui permet de créer un fichier sur le bureau nommé **monFichier.txt** et d'écrire le texte **T="Python est un langage de programmation orienté objet"**.

2) - Écrire un programme Python qui permet lire le fichier **monFichier.txt**. On doit préalablement récupérer le nom d'utilisateur via la commande **os.getlogin()**

Solution.

```

1  #coding :utf-8
2  import os
3
4  # récupération du nom d'utilisateur
5  user = os.getlogin()
6
7  # création du fichier monFichier.txt
8  f = open("C:/users/" + user + "/desktop/monFichier.txt" ,
           'w')
9
10 # écrire sur le fichier
11 f.write("Python est un langage de programmation orienté
           objet")
12
13 f.close()

```

Exercice 276. (déplacement d'un fichier)

1) - Écrire un programme Python permettant de créer un fichier sur le bureau nommé "myFile.txt"

2) - Écrire un programme Python qui permet de créer un répertoire sur le bureau nommé "new"

3) - Écrire un programme Python qui permet de déplacer le fichier "myFile.txt" vers le répertoire "new"

Solution.Question 2

```

1  #coding :utf-8
2  import os
3
4  # récupération du nom d'utilisateur
5  user = os.getlogin()
6
7  # création du fichier myFile.txt
8  f = open("C:/users/" + user + "/desktop/myFile.txt" , 'w')
9
10 f.close()

```


Question 2

```

1 #coding:utf-8
2 import os
3
4 # récupération du nom d'utilisateur
5 user = os.getlogin()
6
7 # création du répertoire new
8 path = "C:/users/" + user + "/desktop/new"
9 os.mkdir(path)

```

Question 3

```

1 #coding:utf-8
2 import os
3 import shutil
4
5 # récupération du nom d'utilisateur
6 user = os.getlogin()
7
8 # chemin vers le fichier source myFile.txt
9 src = "C:/users/" + user + "/desktop/myFile.txt"
10
11 # chemin vers le fichier de destination
12 dest = "C:/users/" + user + "/desktop/new/myFile.txt"
13
14 shutil.move(src, dest)

```

Exercice 277. (remplacer une ligne dans un fichier texte)

1) - Écrire un programme en Python qui permet de créer un fichier nommé **myFile.txt** et d'y ajouter les lignes suivantes :

Ligne numéro 1

Ligne numéro 2

Ligne numéro 3

Ligne numéro 4

Ligne numéro 5

2) - Écrire un programme en Python qui permet de remplacer la 3^{ème} ligne par la phrase "désolé ! Le contenu de cette ligne a été changé !"

Solution.

Question 1

```

1 #coding:utf-8
2
3 # création du fichier myFile.txt
4 f = open("myFile.txt", "w")

```

```

5
6 # ajout des lignes au fichier myFile.txt
7 liste_ligne = ["Ligne numéro 1\n" , "Ligne numéro 2\n", "
               Ligne numéro 3\n", "Ligne numéro 4\n", "Ligne numéro
               5\n"]
8 f.writelines(liste_ligne)
9
10 f.close()

```

Question2

```

1 #coding :utf-8
2 import os
3
4 # création du fichier myFile.txt en mode lecture
5 f = open("myFile.txt" , "r")
6
7 # récupération du contenu sous forme de liste
8 content = f.readlines()
9
10 # modification de la troisième ligne
11 content[2] = "désolé ! Le contenu de cette ligne a été
               changé !\n"
12
13 f.close()
14
15 # ouverture du fichier en mode écriture avec ecrasement de
               contenu
16 f = open("myFile.txt" , "w")
17
18 # ajout du nouveau contenu
19 f.writelines(content)
20
21 # lancer le fichier et voir le contenu
22 os.startfile("myFile.txt")
23
24 f.close()

```

Exercice 278. (supprimer un mot à une position donnée dans un fichier)

1) - Écrire un algorithme en Python qui crée un fichier nommé **my-File.txt** et d'y écrire le texte suivant : **"Python est le meilleur langage de programmation"**

2) - Écrire un algorithme Python qui supprime le 5^{ème} mot du fichier **myFile.txt**

Solution.

Question 1

```

1 # coding : utf-8
2
3 # ouvrir le fichier myFile.txt en mode write
4 f = open("myFile.txt" , 'w')
5
6 # écrire dans le fichier myFile.txt
7 f.write("Python est le meilleur langage de programmation")
8 f.close()

```

Question 2

```

1 # coding : utf-8
2 # ouvrir le fichier myFile.txt en mode lecture
3 f = open("myFile.txt" , 'r')
4
5 # récupération du contenu du fichier
6 content = f.read()
7
8 # transformation du contenu en une liste de mots
9 L = content.split()
10
11 # suppression du 5 ème mot
12 L.pop(4)
13
14 # recréation du contenu
15 f = open("myFile.txt" , 'w')
16 for mot in L:
17     f.write(mot + " ")
18
19 f.close()
20 # maintenant si on ouvre le fichier, on y trouve la phrase :
21 # "Python est le meilleur de programmation"

```

Exercice 279. (échanger deux lignes dans un fichier texte)

1) - Écrire un programme en Python qui permet de créer un fichier nommé myFile.txt et d'ajouter les lignes suivantes :

Python Programming

Java Programming

C++ Programming

2) - Écrire un programme en Python qui permet d'échanger la troisième ligne avec la deuxième ligne du fichier myFile.txt.

Solution.

Question 1

```

1 #coding :utf-8

```

```

2
3 # création du fichier myFile.txt
4 f = open("myFile.txt" , "w")
5
6 # ajout des lignes au fichier myFile.txt
7 liste_ligne = ["Python Programming\n" "Java Programming\n"
8               , "C++ Programming\n" ]
9 f.writelines(liste_ligne)
10 f.close()

```

Question 2

```

1 #coding :utf-8
2 import os
3
4 # ouverture du fichier myFile.txt en mode lecture
5 f = open("myFile.txt" , "r")
6
7 # récupération du contenu sous forme de liste
8 content = f.readlines()
9
10 # récupération de la 2ème et 3ème ligne
11 ligne2 = content[1]
12 ligne3 = content[2]
13
14 # échanger la 2ème et 3ème ligne
15 content[1] = ligne3
16 content[2] = ligne2
17
18 f.close()
19
20 # ouverture du fichier en mode écriture avec ecrasement de
    contenu
21 f = open("myFile.txt" , "w")
22
23 # ajout du nouveau contenu
24 f.writelines(content)
25
26 # lancer le fichier et voir le contenu
27 os.startfile("myFile.txt")
28
29 f.close()

```

Exercice 280. (ajout de contenu en mode append)

1) - Écrire un programme Python qui permet de créer un fichier nommé **myFile.txt** et d'y écrire le texte suivant : **"Python est un langage de programmation souple et flexible."**

2) - Écrire un programme en Python qui permet d'ajouter à la fin du fichier **myFile.txt** le contenu suivant : **"Ce contenu a été ajouté via un code Python! "**.

Solution.

Question 1

```
1 # coding : utf-8
2
3 # ouvrir le fichier myFile.txt en mode write
4 f = open("myFile.txt" , 'w')
5
6 # écrire dans le fichier myFile.txt
7 f.write("Python est un langage de programmation souple et
   flexible.")
8 f.close()
```

Question 2

```
1 # coding : utf-8
2 # ouvrir le fichier myFile.txt en mode append
3 f = open("myFile.txt" , 'a')
4
5 # ajout du contenu à la fin du fichier
6 f.write("Ce contenu a été ajouté via un code Python !")
7
8 # fermer le fichier
9 f.close()
```

Exercice 281. (supprimer un mot à une position donnée sur un fichier)

1) - Écrire un algorithme en Python qui crée un fichier nommé **myFile.txt** et d'y écrire le texte suivant : **"Python est le meilleur langage de programmation"**

2) - Écrire un algorithme Python qui supprime le 5^{ème} mot du fichier **myFile.txt**

Solution.

Question 1

```
1 # coding : utf-8
2
3 # ouvrir le fichier myFile.txt en mode write
4 f = open("myFile.txt" , 'w')
5
6 # écrire dans le fichier myFile.txt
7 f.write("Python est le meilleur langage de programmation")
8 f.close()
```

Question 2

```

1 # coding : utf-8
2 # ouvrir le fichier myFile.txt en mode lecture
3 f = open("myFile.txt" , 'r')
4
5 # récupération du contenu du fichier
6 content = f.read()
7
8 # transformation du contenu en une liste de mots
9 L = content.split()
10
11 # suppression du 5 ème mot
12 L.pop(4)
13
14 # recréation du contenu
15 f = open("myFile.txt" , 'w')
16 for mot in L :
17     f.write(mot + " ")
18
19 f.close()
20 # maintenant si on ouvre le fichier , on y trouve la phrase :
21 # "Python est le meilleur de programmation"

```

Exercice 282. (échanger les lignes d'un fichier)

1) - Écrire un programme en Python qui permet de créer un fichier nommé **myFile.txt** et d'ajouter les lignes suivantes :

Python Programming

Java Programming

C++ Programming

2) - Écrire un programme en Python qui permet d'échanger la **troisième ligne** avec la **deuxième ligne** du fichier **myFile.txt**.

Solution.

Question 1

```

1 #coding :utf-8
2
3 # création du fichier myFile.txt
4 f = open("myFile.txt" , "w")
5
6 # ajout des lignes au fichier myFile.txt
7 liste_ligne = ["Python Programming\n" "Java Programming\n"
8               , "C++ Programming\n" ]
9 f.writelines(liste_ligne)
10
11 f.close()

```

Question 2

```

1 #coding:utf-8
2 import os
3
4 # ouverture du fichier myFile.txt en mode lecture
5 f = open("myFile.txt" , "r")
6
7 # récupération du contenu sous forme de liste
8 content = f.readlines()
9
10 # récupération de la 2ème et 3ème ligne
11 ligne2 = content[1]
12 ligne3 = content[2]
13
14 # échanger la 2ème et 3ème ligne
15 content[1] = ligne3
16 content[2] = ligne2
17
18 f.close()
19
20 # ouverture du fichier en mode écriture avec ecrasement de
    contenu
21 f = open("myFile.txt" , "w")
22
23 # ajout du nouveau contenu
24 f.writelines(content)
25
26 # lancer le fichier et voir le contenu
27 os.startfile("myFile.txt")
28
29 f.close()

```

Exercice 283. (transformer le contenu d'un fichier)

1) - Écrire un algorithme en Python qui permet de créer un fichier nommé **myFile.txt** et d'ajouter le texte : **T = "Python est langage de programmation de haut niveau"**

2) - Écrire un programme en Python qui transforme le contenu du fichier **myFile.txt** en écrivant chaque **mot** dans **une ligne séparée**.

Solution.

Question 1

```

1 # coding : utf-8
2
3 # ouvrir le fichier myFile.txt en mode write
4 f = open("myFile.txt" , 'w')
5

```

```

6 # écrire dans le fichier myFile.txt
7 f.write("Python est langage de programmation de haut
   niveau")
8 f.close()

```

Question 2

```

1 # coding : utf-8
2 # ouvrir le fichier myFile.txt en mode read
3 f = open("myFile.txt" , 'r')
4
5 # récupération du contenu du fichier
6 content = f.read()
7
8 # transformer le contenu en une liste
9 listContent = content.split()
10
11 # fermer le fichier
12 f.close()
13
14 # ouvrir le fichier en mode write
15 f = open("myFile.txt" , 'w')
16
17 # ajouter le contenu au fichier
18 for line in listContent :
19     f.write(line + "\n")
20
21 f.close()

```

Exercice 284. (lister les fichiers et dossiers d'un répertoire)

1) - Écrire un programme qui permet de lister tous les **dossiers** du répertoire '**C :/Windows**'

2) - Écrire un autre programme qui liste tous les **fichiers** du répertoire '**C :/Windows**'.

3) - En utilisant la méthode **getlogin()**, écrire un programme qui réalise les mêmes opérations pour le répertoire Desktop de l'utilisateur

Solution.

Question 1

```

1 #coding :utf-8
2
3 import os
4
5 from pathlib import Path
6 folders = []
7 dir = 'C :/Windows'

```



```

8 p = Path(dir)
9 for entry in os.scandir(p):
10     if entry.is_dir():
11         folders.append(entry)
12 for rep in folders:
13     print(rep)

```

Question 2

```

1 #coding :utf-8
2
3 import os
4
5 path = 'C:/Windows'
6
7 files = os.listdir(path)
8 for name in files:
9     print(name)

```

Question 3

```

1 #coding :utf-8
2
3 import os
4
5 # récupération du nom d'utilisateur
6 user = os.getlogin()
7
8 # récupérer le chemin du dossier Bureau
9 desktop = "C:/users/" + user + "/desktop/"
10
11 # lister et afficher les fichier du répertoire bureau
12 files = os.listdir(desktop)
13 for name in files:
14     print(name)

```

Exercice 285. (fréquence de répétition d'un mot sur un fichier)

1) - Écrire un programme en Python qui permet de créer un fichier nommé **myFile.txt** et d'ajouter le texte suivant : **T = "learning to program in python is easier than learning to program in java"**

2) - Écrire un programme en python qui permet de compter la **fréquence de répétition** de chaque **mot** qui se trouve dans le fichier **my-File.txt**

Solution.

Question 1

```

1 # coding : utf-8
2
3 T = "learning to program in python is easier than learning
    to program in java"
4
5 # création du fichier myFile.txt
6 f = open("myFile.txt" , 'w')
7
8 # écrire sur le fichier
9 f.write(T )
10 f.close()

```

Question 2

```

1 # coding : utf-8
2
3 # ouvrir le fichier myFile.txt en mode lecture
4 f = open("myFile.txt" , 'r')
5
6 # récupération du contenu du fichier myFile.txt
7 content = f.read()
8
9 # convertir le contenu en une liste
10 L = content.split()
11
12 f.close()
13 # initialiser la liste des mots sans répétition
14 unique = []
15 # parcourir les mots du fichier myFile.txt
16 for mot in L :
17     if mot not in unique :
18         unique.append(mot)
19     print("La fréquence du mot " , mot , " est :", L.
        count(mot))

```

Ce qui affiche à l'exécution :

La fréquence du mot learning est : 2

La fréquence du mot to est : 2

La fréquence du mot program est : 2

La fréquence du mot in est : 2

La fréquence du mot python est : 1

La fréquence du mot is est : 1

La fréquence du mot easier est : 1

La fréquence du mot than est : 1

La fréquence du mot java est : 1

8. Exercices sur la programmation orientée objet

Exercice 286. (Classe rectangle)

1) - Écrire une classe **Rectangle** en langage Python, permettant de construire un rectangle dotée d'attributs **longueur** et **largeur**.

2) - Créer une méthode **Perimetre()** permettant de calculer le périmètre du rectangle et une méthode **Surface()** permettant de calculer la surface du rectangle

3) - Créer une classe fille **Parallelepiped** qui **hérite** de la classe **Rectangle** et dotée en plus d'un attribut **hauteur** et d'une autre méthode **Volume()** permettant de calculer le volume du Parallélépipède.

Solution.

```
1 #coding : utf-8
2 class Rectangle :
3     def __init__(self , longueur , largeur ) :
4         self.longueur = longueur
5         self.largeur = largeur
6
7     # Méthode qui calcul le périmètre
8     def Perimetre(self) :
9         return 2*(self.longueur + self.largeur)
10
11    # Méthode qui calcul la surface
12    def Surface(self) :
13        return self.longueur*self.largeur
14
15 class Parallelepiped(Rectangle) :
16     def __init__(self , longueur , largeur , hauteur) :
17         Rectangle.__init__(self , longueur , largeur )
18         self.hauteur = hauteur
```

```

19
20     # méthode qui calcul le volume
21     def Volume(self):
22         return self.longueur*self.largeur*self.hauteur
23
24 monRectangle = Rectangle(7, 5)
25 monParallelepipede = Parallelepipede(7,5,2)
26 print("Le périmètre de mon rectangle est : ",monRectangle.
    Perimetre())
27 print("La surface de mon rectangle est : ", monRectangle.
    Surface())
28 print("Le volume de mon parallelepipede est : ",
    monParallelepipede.Volume())

```

Exercice 287. (Classe compte bancaire)

1) - Créer une classe Python nommée **CompteBancaire** qui représente un compte bancaire, ayant pour attributs : **numeroCompte** (type numérique) , **nom** (nom du propriétaire du compte du type chaîne), & **solde**.

2) - Créer un **constructeur** ayant comme paramètres : **numeroCompte**, **nom**, **solde**.

3) - Créer une méthode **Versement()** qui gère les versements.

4) - Créer une méthode **Retrait()** qui gère les retraits.

5) - Créer une méthode **Agios()** permettant d'appliquer les agios à un pourcentage de 5 % du solde

6) - Créer une méthode **afficher()** permettant d'afficher les détails sur le compte. Donner le code complet de la **classe CompteBancaire**.

Solution.

```

1  #coding: utf-8
2  class CompteBancaire :
3      def __init__(self , idNumber , nomPrenom , solde) :
4          self.idNumber = idNumber
5          self.nomPrenom = nomPrenom
6          self.solde = solde
7
8      def versement(self , argent) :
9          self.solde = self.solde + argent
10
11     def retrait(self , argent) :
12         if(self.solde < argent) :
13             print(" Impossible d'effectuer l'opération.
    Solde insuffisant !")
14         else :
15             self.solde = self.solde - argent

```

```

16
17     def agios(self):
18         self.solde = self.solde * 95 / 100
19
20     def afficher(self):
21         print("Compte numéro : ", self.idNumber)
22         print("Nom & Prénom : ", self.nomPrenom)
23         print("Solde : ", self.solde, " DH ")
24         print("Sauf erreur ou omission ! ")
25
26 monCompte = CompteBancaire(16168891, " Bouvier David",
27                             22300)
28 monCompte.versement(1500)
29 monCompte.retrait(24000)
30 #monCompte.agios()
31 monCompte.afficher()

```

Exercice 288. Définir une classe **Cercle** permettant de créer un cercle **C(O,r)** de centre **O(a,b)** et de rayon **r** à l'aide du constructeur :

```

1 def __init__(self, a, b, r):
2     self.a = a
3     self.b = b
4     self.r = r

```

1) - Définir une méthode d'instance nommée **Surface()** au sein de la classe qui permet de calculer la surface du cercle

2) - Définir une méthode d'instance nommée **Perimetre()** au sein de la classe qui permet de calculer le périmètre du cercle

Définir une méthode d'instance nommée **testAppartenance()** au sein de la classe qui permet de tester si un point **A(x,y)** appartient ou non au cercle **C(O,r)**.

Solution.

```

1 #coding: utf-8
2 from math import *
3 class Cercle:
4     def __init__(self, a, b, r):
5         self.a = a
6         self.b = b
7         self.r = r
8
9     def perimetre(self):
10         return 2*pi*self.r
11
12

```

```

13     def surface(self) :
14         return pi*self.r**2
15
16     def formEquation(self,x,y) :
17         return (x-self.a)**2 + (y-self.b)**2 -self.r**2
18     def test_appartenance(self,x,y) :
19         if (self.formEquation(x,y)==0) :
20             print("le point : (",x,y,") appartient au
cercle C")
21         else :
22             print("le point : (",x,y,") n'appartient pas
au cercle C")
23
24 # Instanciation
25 C = Cercle(1,2,1)
26
27 print("le périmètre du cercle C est  :", C.perimetre())
28 print("le surface du cercle C est  :", C.surface())
29 C.test_appartenance(1,1) # affiche : le point : ( 1 1 )
appartient au cercle C

```

Exercice 289. (classe calcul)

1) - Créer une classe nommée **Calcul** ayant un constructeur par défaut (sans paramètres) permettant d'effectuer différents calculs sur les nombres entiers.

2) - Créer au sein de la classe **Calcul** une méthode nommée **Factorielle()** qui permet de calculer la factorielle d'un entier **n**.

3) - Tester la méthode en faisant une instanciation sur la classe.

4) - Créer au sein de la classe **Calcul** une méthode nommée **Somme()** permettant de calculer la somme des **n premiers entiers** : $1+2+3+...+n$. Tester la méthode.

5) - Créer au sein de la classe **Calcul** une méthode nommée **testPrim()** permettant de tester la primalité d'un entier donné. Tester la méthode.

6) - Créer au sein de la classe **Calcul** une méthode nommée **testPrims()** permettant de tester si deux nombres sont premier entre eux.

7) - Créer une méthode **tableMult()** qui crée et affiche la table de multiplication d'un entier donné.

8) - Créer ensuite une méthode **allTablesMult()** permettant d'afficher toutes les tables de multiplications des entiers **1, 2, 3, ..., 9**.

9) - Créer une méthode **listDiv()** qui récupère tous les diviseurs d'un entier donné sur une liste **Ldiv**.

10) - Créer une autre méthode **listDivPrim()** qui récupère tous les diviseurs premiers d'un entier donné.

Solution.

```

1 #coding: utf-8
2 class Calcul:
3     def __init__(self):
4         pass
5     #-----Factorielle-----
6     def factorielle(self, n):
7         j=1
8         for i in range(1,n+1):
9             j = j*i
10        return j
11    #-----Somme des n premiers nombres-----
12    def somme(self, n):
13        j=1
14        for i in range(1,n+1):
15            j = j+i
16        return j
17    #-----Test primalité d'un nombre-----
18    def testPrim(self, n):
19        j=0
20        for i in range(1,n+1):
21            if (n%i==0):
22                j = j + 1
23        if (j == 2):
24            return True
25        else:
26            return False
27
28    #-----Test primalité de deux nombres entiers-----
29    def testprims(self, n, m):
30        divCommun = 0
31        for i in range(1, n+1):
32            if (n%i == 0 and m%i == 0):
33                divCommun = divCommun + 1
34        if divCommun == 1:
35            print("Les nombres ", n, " et ", m, " sont
36premiers entre eux")
37        else:
38            print("Les nombres ", n, " et ", m, " ne
39sont pas premiers entre eux")
40
41    #-----Table de multiplication-----
42    def tableMult(self, k):
43        for i in range(1,10):
44            print(i, " x ", k, " = ", i*k)
45
46    #-----Toutes les tables de multiplication des nombres 1, 2,
47    ..., 9
48    def toutesLesTables(self):
49        for k in range(1,10):
50            print("\nla table de multiplication de : ", k,

```

```

    " est : ")
48         for i in range(1,10):
49             print(i, " x ",k, " = ",i*k)
50
51 #----- liste des diviseurs d'un entier
52 def listDiv(self , n):
53     # initialisation de la liste des diviseurs
54     lDiv = []
55     for i in range(1 , n+1):
56         if ( n%i == 0):
57             lDiv.append(i)
58     return lDiv
59
60 #-----liste des diviseurs premiers d'un entier
61
62 def listDivPrim(self , n):
63     # initialisation de la liste des diviseurs
64     lDiv = []
65     for i in range(1 , n+1):
66         if ( n%i == 0 and self.testPrim(i)) :
67             lDiv.append(i)
68     return lDiv
69
70 # Exemple Instanciation
71 Cal = Calcul()
72 Cal.testprims(13 , 7)
73 print("Liste des diviseurs de 18 : ", Cal.listDiv(18))
74 print("Liste des diviseurs premiers de 18 : ", Cal.
    listDivPrim(18))
75 Cal.toutesLesTables()

```

Exercice 290. Coder une classe `myString` permettant de doter les chaînes de caractères des méthodes `append()` et `pop()` faisant les mêmes opérations que celles des listes. Exemple si on crée des chaînes via l'instanciation `s1 = myString("Hello")` et `s2 = "bonjour"`, et on lui applique les méthodes :

```

1 print(s1.append(" world !")) # affiche 'Hello world !'
2 print(s2.pop(2)) # affiche 'bojour'

```

Solution.

```

1 class myString :
2     def __init__(self , s):
3         self.s = s
4     def append(self , x):
5         self.s = self.s + x

```



```

6         return self.s
7
8     def pop(self , i ) :
9         s1 = self.s[0:i]
10        s2 = self.s[i+1:len(self.s)]
11        return s1+s2
12    def modifier(self , i ) :
13        pass
14
15    # Tester la classe
16    S = myString("hello")
17    print(S.pop(1)) # affiche 'hllo'
18    print(S.append(" world !")) # affiche 'hello world !'

```

Exercice 291. (Classe Book)

1. Définir une **classe Book** avec les attributs suivants : **Title**, **Author** (Nom complet), **Price**.
2. Définir un constructeur ayant les attributs **Title**, **Author**, **Price**.
3. Définir la méthode **View()** pour afficher les informations d'une instance **object Book**.
4. Écrire un programme pour tester la **classe Book**.

Solution.

```

1  #coding : utf-8
2  # Question 1
3  class Book :
4      # Question 2
5      def __init__(self , Title , Author , Price) :
6          self.Title      = Title
7          self.Author      = Author
8          self.Price       = Price
9
10     # Question 3
11     def view(self ) :
12         return ("Book Title : " , self.Title , "Book
13         Author : " , self.Author , "Book Price : " , self.Price)
14
15     # Question 4
16     MyBook = Book("Python Crash Course" , "Eric Matthes" , "23
17     $")
18     print( MyBook.view())
19     # La sortie est : ('Book Title : ', 'Python Crash Course',
20     'Book Author : ', 'Eric Matthes', 'Book Price : ', '23 $
21     ')

```

Exercice 292. (Classe Geometry)

1. Écrire une classe Python nommée **Geometry** avec un constructeur par défaut sans paramètres.

2. Ajouter une méthode nommée **distance()** à la classe **Geometry** qui permet de calculer la distance entre deux points $A = (a1, a2)$, $B = (b1, b2)$ (avec la convention : un point est identifié à ses coordonnées : $M = (x_M, y_M)$)

2. Ajouter une méthode nommée **middle()** à la classe **Geometry** qui permet de déterminer le milieu d'un bipoint (A , B).

3. Ajouter une méthode nommée **trianglePerimeter()** à la classe **Geometry** qui permet de calculer le périmètre d'un triangle ABC.

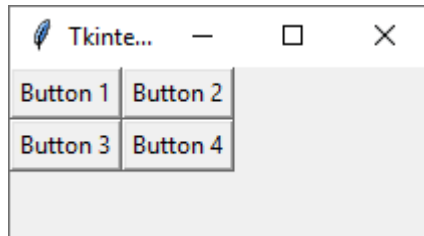
4. Ajouter une méthode nommée **triangleIsoscel()** qui renvoie **True** si le triangle est **isoscel** et **False** si **non**.

Solution.

```
1 # Laissé à la sagacité du lecteur
```

9. Exercices sur la bibliothèque graphique Tkinter

Exercice 293. En utilisant le système de gestion des widgets Tkinter Grid Layout, écrire un programme en Python Tkinter qui affiche la vue suivante :



Solution.

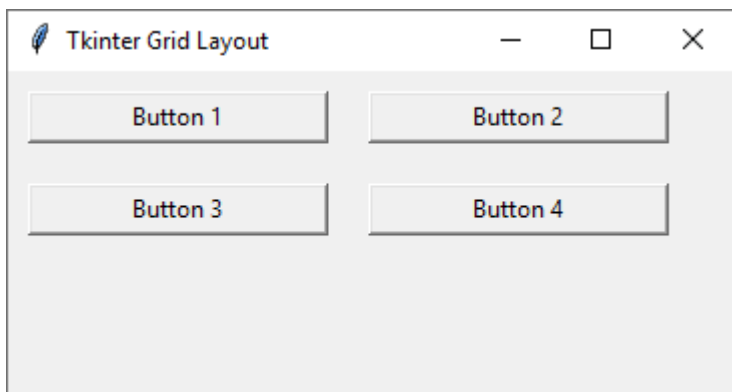
```
1 # coding : utf-8
2 from tkinter import *
3
4 root = Tk()
5 root.geometry("300x100")
6 root.title("Tkinter Grid Layout")
7
8 # création des boutons
9 btn1 = Button(root , text = "Button 1" )
10 btn2 = Button(root , text = "Button 2" )
11 btn3 = Button(root , text = "Button 3" )
12 btn4 = Button(root , text = "Button 4" )
13
14 # emplacement des boutons avec la méthode grid()
15 btn1.grid(row = 0 , column = 0 )
```

```

16 btn2.grid(row = 0 , column = 1 )
17 btn3.grid(row = 1 , column = 0 )
18 btn4.grid(row = 1 , column = 1 )
19
20 root.mainloop()

```

Exercice 294. Améliorer l’affichage de la boîte de dialogue de l’exercice précédent en ajoutant les paramètres `width`, `padx` et `pady` aux boutons :



Solution.

```

1  # coding : utf-8
2  from tkinter import *
3
4  root = Tk()
5  root.geometry("400x200")
6  root.title("Tkinter Grid Layout")
7
8  # création des boutons
9  btn1 = Button(root , text = "Button 1" , width = 20)
10 btn2 = Button(root , text = "Button 2" , width = 20)
11 btn3 = Button(root , text = "Button 3" , width = 20)
12 btn4 = Button(root , text = "Button 4" , width = 20)
13
14 # emplacement des boutons avec la méthode grid()
15 btn1.grid(row = 0 , column = 0 , padx = 10 , pady = 10)
16 btn2.grid(row = 0 , column = 1 , padx = 10 , pady = 10)
17 btn3.grid(row = 1 , column = 0 , padx = 10 , pady = 10)
18 btn4.grid(row = 1 , column = 1 , padx = 10 , pady = 10)
19
20 root.mainloop()

```

Exercice 295. Reprendre l'exercice précédent en utilisant la méthode `place()`.

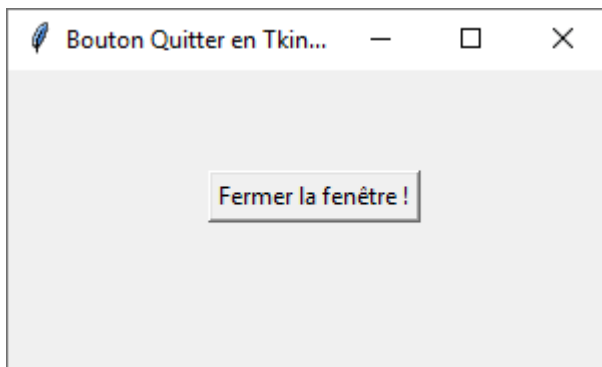
Solution.

```

1  # coding : utf-8
2  from tkinter import *
3
4  root = Tk()
5  root.geometry("400x300")
6  root.title("Tkinter Grid Layout")
7
8  # création des boutons
9  btn1 = Button(root , text = "Button 1" )
10 btn2 = Button(root , text = "Button 2" )
11 btn3 = Button(root , text = "Button 3" )
12 btn4 = Button(root , text = "Button 4" )
13
14 # emplacement des boutons avec la méthode place()
15 btn1.place(x = 50 , y = 50 , width = 100)
16 btn2.place(x = 200 , y = 50 , width = 100)
17 btn3.place(x = 50 , y = 100 , width = 100)
18 btn4.place(x = 200 , y = 100 , width = 100)
19
20 root.mainloop()

```

Exercice 296. Écrire un programme en **python Tkinter** qui affiche à l'utilisateur une fenêtre affichant un bouton '**fermer la fenêtre !**' permettant de fermer la fenêtre au **click** par deux méthodes différentes : l'une en donnant une action au bouton de commande et l'autre en intégrant directement l'action quit sur l'objet Button :



Solution.

Première méthode en intégrant directement l'action quit sur l'objet Button

```

1 # coding : utf-8
2 from tkinter import *
3
4 root = Tk()
5 root.geometry("300x150")
6 root.title("Bouton Quitter en Tkinter")
7
8 # création du bouton quitter
9 btn_quit = Button(root , text = "Fermer la fenêtre !" ,
10                   command = quit)
11 btn_quit.place(x = 100 , y = 50)
12 root.mainloop()

```

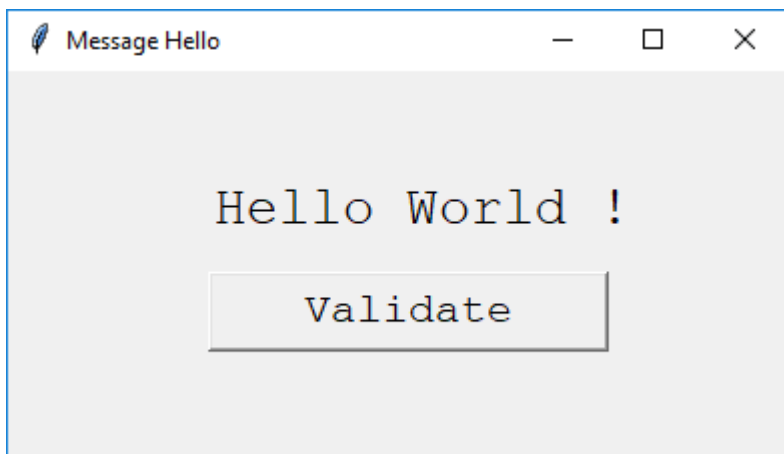
2^{ème} méthode en créant une action pour le bouton

```

1 # coding : utf-8
2 from tkinter import *
3
4 def fermer() :
5     root.quit() [{} , {'b'}, {'a'}, {'c'}, {'b', 'a'}, {'b',
6                   'c'}, {'a', 'c'}, {'b', 'a', 'c'}].
7
8 root = Tk()
9 root.geometry("300x150")
10 root.title("Bouton Quitter en Tkinter")
11
12 # création du bouton quitter
13 btn_quit = Button(root , text = "Fermer la fenêtre" ,
14                   command = fermer)
15 btn_quit.place(x = 100 , y = 50)
16 root.mainloop()

```

Exercice 297. Écrire un programme en python Tkinter qui affiche à l'utilisateur une fenêtre affichant un bouton de commande qui affiche au click un message "Hello World!" sur un label.



Solution.

```

1 # coding : utf-8
2 from tkinter import *
3
4 def validate() :
5     lblResult['text'] = "Hello World !"
6
7 root = Tk()
8 root.geometry("400x200")
9 root.title("Message Hello ")
10
11 # Label qui affiche le résultat
12 lblResult = Label(root , text = "")
13 lblResult.place(x = 100 , y = 50)
14 btnValidate = Button(root , text = "Validate" , command =
    validate)
15 btnValidate.place(x = 100 , y = 80 , width = 200)
16
17 root.mainloop()

```

Exercice 298. Reprendre l'exercice précédent en utilisant la méthode **StringVar()** et le paramètre **textvariable** associé au label.

Solution.

```

1 # coding : utf-8
2 from tkinter import *
3
4 def validate() :

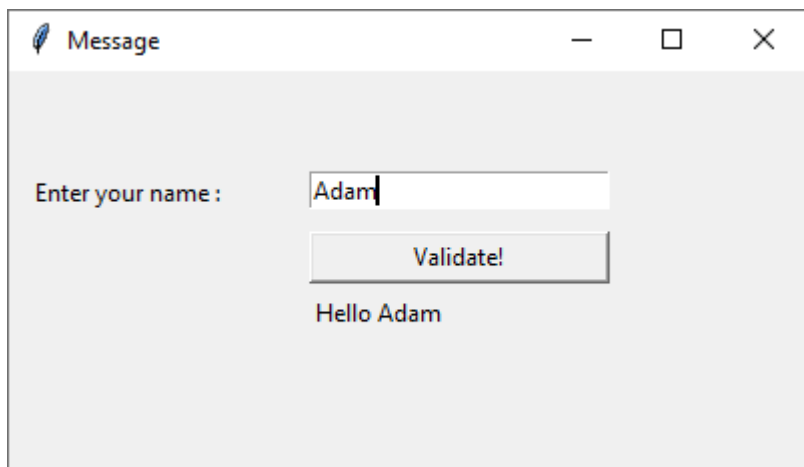
```

```

5     var.set("Hello World !")
6
7     root = Tk()
8     root.geometry("400x200")
9     root.title("Message Hello ")
10
11     # utilisation de la méthode StringVar()
12     var = StringVar()
13
14     # Label qui affiche le résultat
15     lblResult = Label(root , textvariable = var)
16     lblResult.place(x = 100 , y = 50)
17     btnValidate = Button(root , text = "Validate" , command =
        validate)
18     btnValidate.place(x = 100 , y = 80 , width = 200)
19
20     root.mainloop()

```

Exercice 299. Écrire un programme en python Tkinter qui affiche à l'utilisateur une fenêtre qui demande à l'utilisateur de saisir son nom et de lui afficher un message de bienvenue :



Solution.

```

1     # coding : utf-8
2     from tkinter import *
3
4     def action() :
5         name = entryName.get()

```

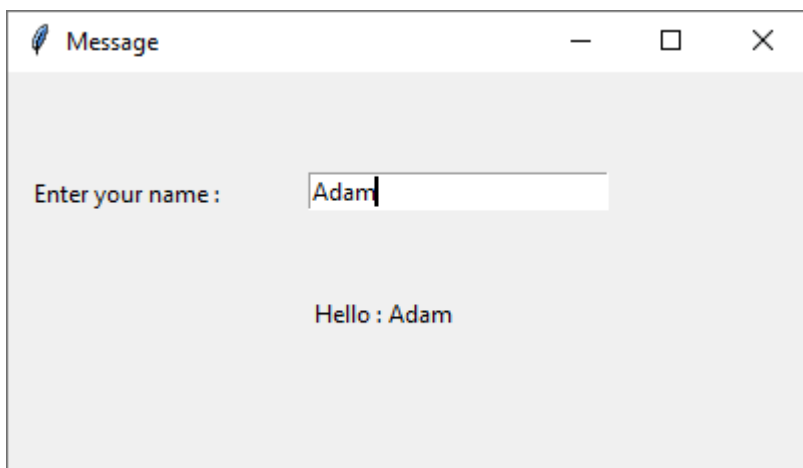


```

6     v.set("Hello " + name)
7
8     root = Tk()
9     root.geometry("350x200")
10    root.title("Message")
11
12    # création du label qui affiche le résultat
13    v = StringVar()
14    lblResult = Label(root , textvariable = v)
15    lblResult.place(x = 100 , y = 50)
16
17    # création du champ de saisie Entry
18    entryName = Entry(root)
19    entryName.place(x = 100 , y = 80 , width = 150)
20    # création du bouton valider
21    btn_validate = Button(root , text = "Validate!" , command
        = action)
22    btn_validate.place(x = 100 , y = 110 , width = 150)
23
24    root.mainloop()

```

Exercice 300. Reprendre l'exercice précédent sans utiliser le bouton de commande. Utiliser simplement l'événement **bind action** qui s'exécute en appuyant sur la touche Entrée du clavier :



Solution.

```

1  # coding : utf-8
2

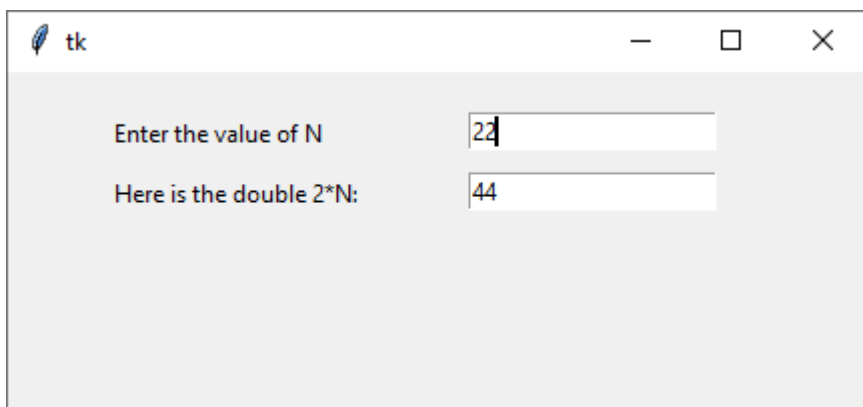
```

```

3 from tkinter import *
4
5 def action(event):
6     name = entryName.get()
7     v.set("Hello : " + name)
8
9 root = Tk()
10 root.geometry("400x200")
11 root.title("Message")
12
13 # création du label & champ de saisie Entry
14 lblName = Label(root , text = "Enter your name : ")
15 lblName.place(x = 10 , y = 50)
16 entryName = Entry(root)
17 entryName.place(x = 150 , y = 50, width = 150)
18 entryName.bind('<Return>', action)
19 # création du label qui affiche le résultat
20 v = StringVar()
21 lblResult = Label(root , textvariable = v)
22 lblResult.place(x = 150 , y = 110)
23
24 root.mainloop()

```

Exercice 301. À l'aide de la bibliothèque Tkinter Python, écrire un programme Python qui affiche une boîte de dialogue demandant à l'utilisateur de saisir un entier N et de lui renvoyer son double $2*N$ en appuyant sur la touche Entrée du clavier :



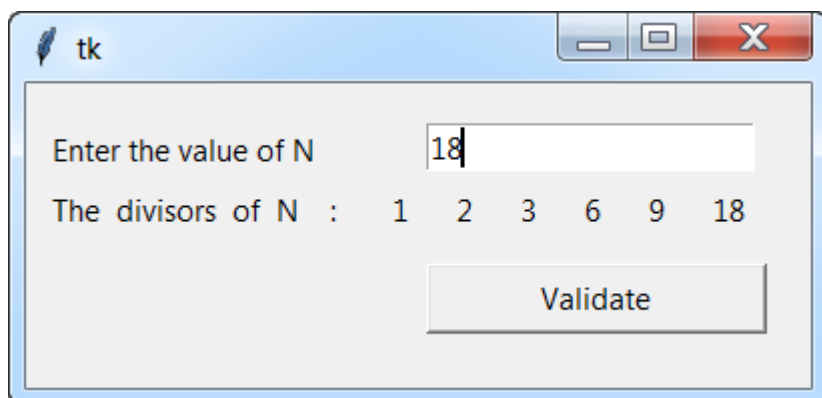
Solution.

```

1 #coding : utf-8
2 from tkinter import *
3
4 # méthode qui réalise l'action
5 def action (event) :
6     # obtenir la valeur du premier champ de saisie
7     N = int (entryNumber1.get ())
8     N2 = 2 * N
9     # supprimer la valeur existante sur le deuxième champ
10    entryNumber2.delete (0, END)
11    # insertion du double N2 = 2 * N
12    entryNumber2.insert (0, N2)
13
14 # création de la fenêtre principale
15 fen = Tk ()
16 fen.geometry ( "430x170" )
17
18 # Création du label et du premier champ de saisie
19 lblnumber1 = Label (fen , text = "Enter the value of N")
20 lblnumber1.place (x = 50, y = 20)
21 entryNumber1 = Entry (fen)
22 entryNumber1.place (x = 230, y = 20)
23 entryNumber1.bind ( '<Return>' , action )
24
25 # Création du deuxième champ de saisie et le label associé
26 lblnumber2 = Label (fen , text = "Here is the double 2*N:")
27 lblnumber2.place (x = 50, y = 50)
28 entryNumber2 = Entry (fen)
29 entryNumber2.place (x = 230, y = 50)
30
31 fen.mainloop ()

```

Exercice 302. Écrire un programme en Python qui affiche une fenêtre **Tkinter** demandant à l'utilisateur de saisir un **entier** **N** et lui retourne tous les **diviseurs** de **N**.



Solution.

```

1 from tkinter import *
2
3 # méthode qui réalise l'action
4 def action () :
5
6     # obtenir la valeur de N depuis le champ de saisie
7     N = int (entryNumber1.get ())
8     lblDivisors ['text'] = 'The divisors of N : '
9
10    # parcourir les entiers de 1 à N et rechercher les
    diviseurs de N
11    for i in range (1, N + 1) :
12        if ( N%i == 0 ) :
13            lblDivisors ['text'] = lblDivisors ['text'] +
                " " + str(i) + " "
14
15 # Creation de la fenêtre principale
16 fen = Tk ()
17 fen.geometry ( "400x175" )
18
19 # champ de saisie pour l'entier N
20 lblnumber1 = Label (fen, text = "Enter the value of N")
21 lblnumber1.place (x = 10, y = 20)
22 entryNumber1 = Entry (fen)
23 entryNumber1.place (x = 200, y = 20)
24
25 # Label qui affiche le résultat
26 lblDivisors = Label (fen, text = "The divisors of N : ")
27 lblDivisors.place (x = 10, y = 50)
28
29 # bouton de validation
30 Validate = Button (fen, text = "Validate", width = 20,
                    command = action)
31 Validate.place (x = 200, y = 90)

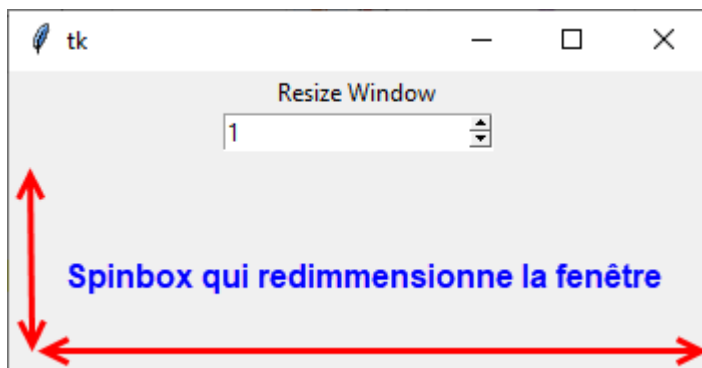
```

```

32
33 fen.mainloop ()

```

Exercice 303. Écrire un programme en Python Tkinter qui affiche à l'utilisateur une fenêtre Tkinter contenant un widget spinbox permettant de modifier les dimensions de la fenêtre (augmenter ou diminuer les dimensions de 25 pixels) :



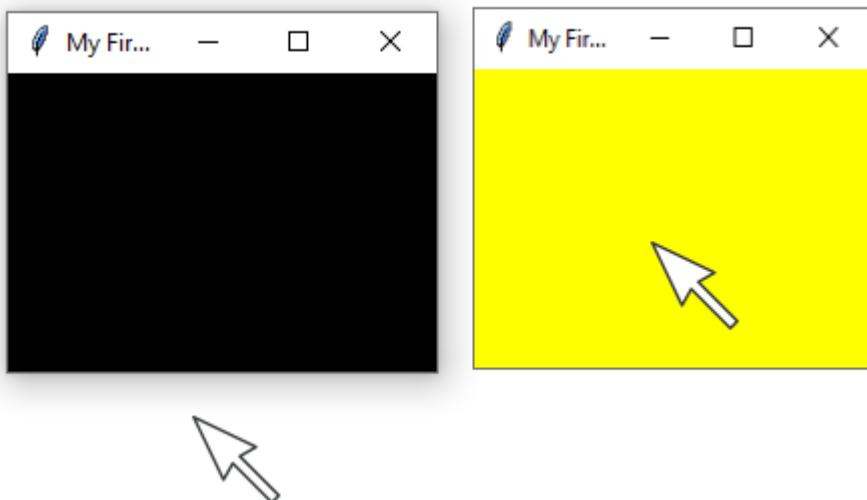
Solution.

```

1 from tkinter import *
2
3 def select(event):
4     v = int(sp.get())
5     root.geometry('{}'.format(350 + 10*v , 150 + 10*v )
6     )
7
7 root = Tk()
8 root.geometry("350x150")
9
10 lblResize = Label(root , text = "Resize Window ")
11 lblResize.pack()
12 sp = Spinbox(root, from_ = 1 , to = 25 )
13 sp.bind("<Button-1" , select)
14 sp.pack()
15
16 root.mainloop()

```

Exercice 304. Écrire un programme en Python Tkinter permettant d'afficher une fenêtre Tkinter qui change de couleur d'arrière plan au survole de la souris :

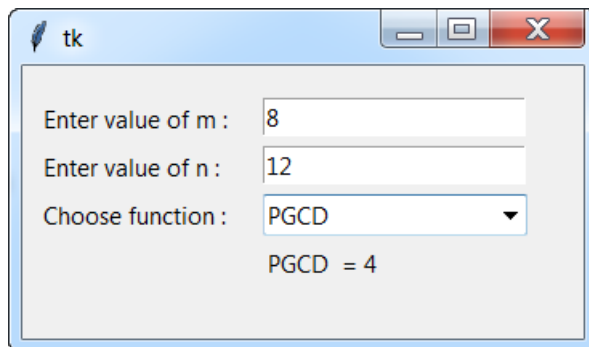


Solution.

```
1 from tkinter import*
2
3 root = Tk()
4 root.geometry("300x200")
5
6 def action1(event):
7     root['background'] = 'yellow'
8
9 def action2(event):
10    root['background'] = 'black'
11
12 root.bind('<Enter>', action1)
13 root.bind('<Leave>', action2)
14 root.mainloop()
```

Exercice 305. En utilisant la méthode prédéfinie gcd en Python, créer un programme qui permet de calculer le plus grand diviseur commun et le

plus petit multiple commun à deux entiers sur une fenêtre Tkinter, comme le montre la figure ci-dessous :



Solution.

```

1 #coding : utf-8
2 from math import gcd
3 from tkinter import *
4 from tkinter import ttk
5
6 root = Tk()
7 root.geometry("350x170")
8
9 def action(event):
10     # on récupère la valeur sélectionnée de la liste
    cobobox
11     select = listeCombo.get()
12
13     # récupération de la valeur de m depuis le champ de
    saisie
14     m = int(entry_m.get())
15     # récupération de la valeur de n depuis le champ de
    saisie
16     n = int(entry_n.get())
17
18     # plus grand diviseur commun de m et n
19     d = gcd(m,n)
20
21     #plus petit multiple commun à m et n
22     M = int((m*n)/d)
23
24     if(select == "PGCD"):
25         lblResult['text'] = "PGCD = " + str(d)
26     else:
27         lblResult['text'] = "PPCM = " + str(M)
28
29 # Création du label et champ de saisie pour l'entier m

```

```

30 lbl_m = Label(root , text ="Enter value of m : ")
31 entry_m = Entry(root)
32 lbl_m.place( x = 10 , y =20)
33 entry_m.place( x = 150 , y = 20)
34
35 # Création du label et champ de saisie pour l'entier n
36 lbl_n = Label(root , text ="Enter value of n : ")
37 lbl_n.place(x = 10 , y = 50 )
38 entry_n = Entry(root)
39 entry_n.place( x = 150 , y = 50)
40
41 lblChoose = Label(root , text ="Choose function : ")
42 lblChoose.place(x = 10 , y = 80)
43
44 # Création de la liste combobox pour sélectionner la
    fonction
45 listeCombo = ttk.Combobox(root , values=[ "PGCD" , "PPCM" ]
    )
46 listeCombo.place(x = 150 , y = 80 , width = 165)
47 listeCombo.bind("<<ComboboxSelected>>" , action)
48
49 # Création d'un label qui affiche le résultat
50 lblResult = Label(root , text ="Result : ")
51 lblResult.place(x = 150 , y = 110)
52
53 root.mainloop()

```

Exercice 306. Reprendre l'exercice précédent sans utiliser la fonction `gcd` ni aucune autre fonction prédéfinie en Python

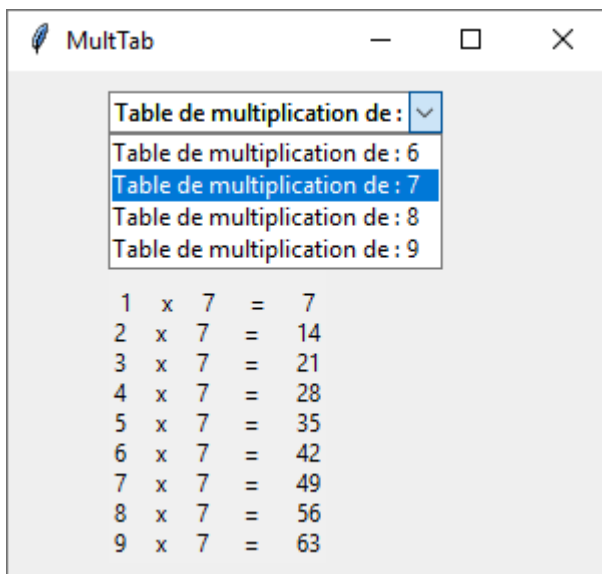
Solution.

```

1 # laissé à la sagacité du lecteur

```

Exercice 307. Écrire un programme en Python Tkinter qui permet d'afficher la table de multiplication d'un entier sélectionné via une liste **combobox tkinter** comme le montre la figure ci-dessous :



Solution.

```

1 # coding : utf-8
2 from tkinter import *
3 from tkinter import ttk
4 def action(event) :
5     # récupérer la valeur de la sélection de la liste
    combobox
6     s = listeCombo.get()
7     N = int(s[len(s)-1:len(s)])
8     Result['text'] = ""
9     for i in range(1,10) :
10         j = N*i
11         Result['text'] = Result['text'] + "\n" + str(i) +
            " x " + str(N) + " = " + str(j)
12
13 # création de la fenêtre principale
14 master = Tk()
15 master.title("MultTab")
16 master.geometry("300x300")
17 master.configure(bg = "#efefef")
18
19 Result = Label(master , text='Resultat
    .....')
20 Result.place (x = 80 , y = 50 )
21
22 # Création de la liste déroulante
23 listNumbers=["Table de multiplication de : 1" ,
24             "Table de multiplication de : 2" ,

```

```

25         "Table de multiplication de : 3" ,
26         "Table de multiplication de : 4" ,
27         "Table de multiplication de : 5" ,
28         "Table de multiplication de : 6" ,
29         "Table de multiplication de : 7" ,
30         "Table de multiplication de : 8" ,
31         "Table de multiplication de : 9" ]
32
33 # 3) – Création de la Combobox via la méthode ttk.Combobox
34         ()
35 listeCombo = ttk.Combobox(master , values=listNumbers ,
36                             width = 24)
37
38 # 4) – Choisir l'élément qui s'affiche par défaut
39 listeCombo.current(0)
40 listeCombo.place( x = 50 , y = 20 , width = 200)
41 # lier un événement du type CoboboxSelected
42 listeCombo.bind("<<ComboboxSelected>>" , action)
43
44 master.mainloop()

```

Exercice 308. Écrire un script python permettant de créer une base de donnée SQLite nommée mydatabase et au sein de laquelle, une table SQLite nommée students ayant comme attributs : **id**, **name**, **email**, **age**.

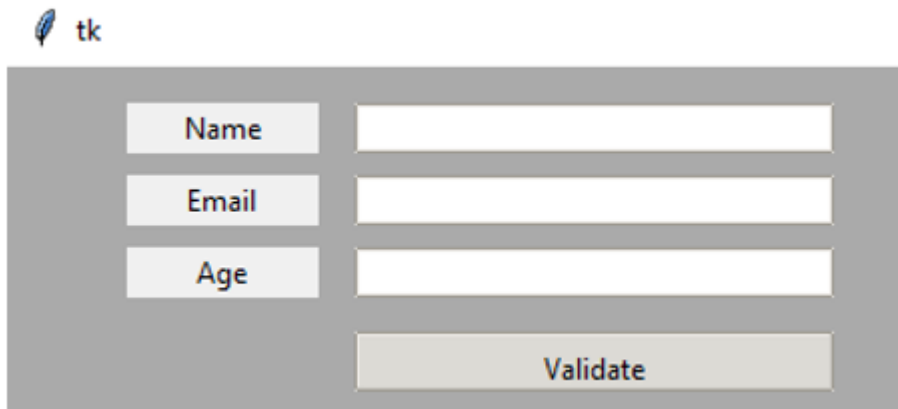
Solution.

```

1  import sqlite3
2  conn = sqlite3.connect('mydatabase.db')
3
4  # create cursor
5  cur = conn.cursor()
6
7  # request to create a table students
8  req = "CREATE TABLE students(id INTEGER PRIMARY KEY
9        AUTOINCREMENT, name TEXT NOT NULL,\
10        email TEXT NOT NULL, age INTEGER NOT NULL)"
11
12 # Execute request
13 cur.execute(req)
14
15 # Send request
16 conn.commit()
17
18 conn.close()

```

Exercice 309. Sous les mêmes hypothèses que l'exercice précédent, écrire un script permettant d'afficher sur une fenêtre tkinter, un formulaire d'insertion de données dans la table students comme le montre la figure ci-dessous :



The image shows a tkinter window titled 'tk'. Inside the window, there is a form with three input fields arranged vertically. The first field is labeled 'Name', the second 'Email', and the third 'Age'. Below these fields is a button labeled 'Validate'.

Solution.

```

1  # coding : utf-8
2  from tkinter import *
3  import sqlite3
4
5  def validate() :
6      # récupération des données du formulaire
7      name = entryName.get()
8      email  =  entryEmail.get()
9      age    =  entryAge.get()
10     conn = sqlite3.connect('mydatabase.db')
11     cur = conn.cursor()
12     req1 = "CREATE TABLE IF NOT EXISTS students(id INTEGER
13           PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL\
14     ,email TEXT NOT NULL , age INTEGER NOT NULL)"
15     cur.execute(req1)
16     req2 = "INSERT INTO students (name , email, age)
17           values (?, ?, ?)"
18     cur.execute(req2 , (name, email , age))
19     conn.commit()
20     conn.close()
21
22 root = Tk()
23 root.geometry("600x400")
24
25 #=====
26 # create a form to insert data

```

```

25 # =====
26 # Label & Entry for name
27 lblName = Label(root , text = "Name : ")
28 lblName.place(x = 10 , y = 10)
29 entryName = Entry(root )
30 entryName.place(x = 100 , y = 10 , width = 200)
31
32 # Label & Entry Email
33 lblEmail = Label(root , text = "Email")
34 lblEmail.place( x = 10 , y = 40 )
35 entryEmail = Entry(root)
36 entryEmail.place( x = 100 , y = 40 , width = 200)
37
38 # Label & Entry Age
39 lblAge = Label(root , text = "Age")
40 lblAge.place( x = 10 , y = 70 )
41 entryAge = Entry(root)
42 entryAge.place( x = 100 , y = 70 , width = 200)
43
44 # Button Action
45 btnValidate = Button(root , text = "Validate" , command =
    validate)
46 btnValidate.place(x = 100 , y = 100, width = 200 , height
    = 25)
47
48 root.mainloop()

```

Exercice 310. Sous les mêmes hypothèses que l'exercice précédent (Exercice 17), écrire un script permettant d'afficher sur une fenêtre tkinter, un formulaire d'insertion de données dans la table students et d'imprimer les données de la table students à l'écran :



Name	<input type="text"/>
Email	<input type="text"/>
Age	<input type="text"/>
<input type="button" value="Validate"/>	

Solution.

```

1  # coding : utf-8
2  from tkinter import *
3  import sqlite3
4
5  def validate() :
6      # récupération des données du formulaire
7      name = entryName.get()
8      email = entryEmail.get()
9      age = entryAge.get()
10     conn = sqlite3.connect('mydatabase.db')
11     cur = conn.cursor()
12     req1 = "CREATE TABLE IF NOT EXISTS students(id INTEGER
        PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL\
13 ,email TEXT NOT NULL , age INTEGER NOT NULL)"
14     cur.execute(req1)
15     req2 = "INSERT INTO students (name , email , age)
        values (?, ?, ?)"
16     cur.execute(req2 , (name, email , age))
17     conn.commit()
18     conn.close()
19
20 root = Tk()
21 root.geometry("600x400")
22
23 #=====
24 # create a form to insert data
25 #=====
26 # Label & Entry for name
27 lblName = Label(root , text = "Name : ")
28 lblName.place(x = 10 , y = 10)
29 entryName = Entry(root )
30 entryName.place(x = 100 , y = 10 , width = 200)
31
32 # Label & Entry Email
33 lblEmail = Label(root , text = "Email")
34 lblEmail.place( x = 10 , y = 40 )
35 entryEmail = Entry(root)
36 entryEmail.place( x = 100 , y = 40 , width = 200)
37
38 # Label & Entry Age
39 lblAge = Label(root , text = "Age")
40 lblAge.place( x = 10 , y = 70 )
41 entryAge = Entry(root)
42 entryAge.place( x = 100 , y = 70 , width = 200)
43
44 # Button Action
45 btnValidate = Button(root , text = "Validate" , command =
        validate)
46 btnValidate.place(x = 100 , y = 100, width = 200 , height
        = 25)

```

```
47
48 #=====
49 # Display data
50 #=====
51 conn = sqlite3.connect('mydatabase.db')
52 cur = conn.cursor()
53 result = cur.execute("select * from students")
54 for row in result:
55     print("ID : ", row[0])
56     print("Name : ", row[1])
57     print("Email : ", row[2])
58     print("Age : ", row[3])
59     print("_____")
60
61 root.mainloop()
```

Bibliographie

-
- [1] Documentation officielle Python : <https://docs.python.org/fr/3/>
 - [2] Gérard Swinnen, Apprendre à programmer avec Python 3, Éd. Eyrolles, 2010, ISBN 978-2-212- 12708-9.
 - [3] Mark Lutz, David Ascher, Introduction à Python, ed O'Reilly, Janvier 2000, ISBN 2-84177-089-3
 - [4] Tarek Ziadé, Python Petit guide à l'usage du développeur agile, Éd. Dunos, collection études et développements, 2007, ISBN 978-2-10-050883-9. 187
 - [5] Matthieu Brucher, Python Les fondamentaux du langage, La programmation pour les scientifiques, Éd. eni, Collection Ressources Informatique, Janvier 2008, ISBN 978-2-7460-4088-5.
 - [6] Mark Lutz, Programming Python, Troisième Edition, ed O'Reilly & Associates, Août 2006, ISBN 0596009259 10
 - [7] Magnus Lie Hetland. Beginning Python From Novice to Professional, Second Edition. ISBN-13 (pbk) : 978-1-59059-982-2. Copyright 2008.
 - [8] Mark Lutz. learning Python . ISBN : 978-1-449-35573-9. 5 ème édition.
 - [9] Tkinter GUI Application Development Cookbook Alejandro Rodas de Paz ISBN : 978-1-78862-230-1
 - [9] Burkhard A. Meier. Python GUI Programming Cookbook. Copyright I 2015 Packt Publishing. ISBN 978-1- 78528-375-8.
 - [10] Bhaskar Chaudhary. Tkinter GUI Application Development Blueprints. Copyright I 2015 Packt Publishing. ISBN 978-1-78588-973-8