



Programmation en langage Python

Younes Derfoufi

► To cite this version:

| Younes Derfoufi. Programmation en langage Python. 2019. hal-02126596v2

HAL Id: hal-02126596

<https://hal.archives-ouvertes.fr/hal-02126596v2>

Preprint submitted on 26 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Programmation en langage Python

Younes. Derfoufi Enseignant au CRMEF OUJDA

21 juin 2019

Table des matières

1 Algorithmique en langage Python	6
1.1 Introduction	6
1.1.1 A propos du langage Python	6
1.1.2 Quelles sont les principales raisons qui poussent à apprendre Python ?	7
1.2 Installation des outils et premier programme Python	8
1.3 Les variables et commentaires en Python	8
1.3.1 Les commentaires en Python	8
1.3.2 les variables en Python	8
1.3.3 Affichage d'une Variable	9
1.4 Les fonctions en Python	9
1.5 Structures de contrôles	9
1.5.1 La structure sélective If ... Else ...	9
1.5.2 La structure répétitive For ...	10
1.5.3 La stucture répétitive While	10
1.6 Les chaines de caractères en Python	11
1.6.1 Définir une chaine de caractère en Python	11
1.6.2 Les fonctions de chaines de caractères en Python	11
1.7 Les listes en Python	14
1.7.1 Création d'une liste en Python	14
1.7.2 Accès aux éléments d'une liste.	14
1.7.3 Changer la valeur d'un élément de la liste	14
1.7.4 Parcourir les éléments d'une liste Python	15
1.7.5 Longueur d'une liste Python	15
1.7.6 Ajouter ou supprimer des éléments à la liste	15
1.7.6.1 Ajouter un un élément à une liste Python	15
1.7.6.2 Retirer un élément d'une liste Python	15
1.7.6.3 Transformer une chaine de caractères en une liste	16
1.7.6.4 Transformer une liste en une chaine de caractères	17
1.7.6.5 Compter la fréquence des éléments d'une liste	17
1.7.7 Les différente méthodes destinées aux listes Python	17
1.8 Les tuples	18
1.8.1 Définir un tuple en Python	18
1.8.2 Accéder aux élément d'un tuple	18

1.8.3 Boucle à travers un tuple	18
1.8.4 Vérifier si un élément existe dans un tuple	18
1.8.5 Longueur d'un tuple	19
1.8.6 Ajout ou suppression d'éléments impossible à un tuple	19
1.8.7 Suppression d'un tuple	19
1.8.8 Création d'un tuple en utilisant le constructeur tuple()	19
1.8.9 Méthodes associées à un tuple	19
1.9 Les dictionnaires	20
1.9.1 Définir un dictionnaire en Python	20
1.9.2 Parcourir les valeurs et les clés d'un dictionnaire Python	20
1.9.3 Mettre à jour, ajouter ou supprimer des éléments d'un dictionnaire	21
1.9.3.1 Mettre à jour un élément du dictionnaire	21
1.9.3.2 Ajouter un élément au dictionnaire	21
1.9.3.3 Supprimer un élément du dictionnaire	21
1.9.3.4 Vider un dictionnaire	22
1.9.4 Récapitulatif des méthodes associées à un dictionnaire	22
1.10 Les ensembles Python (Python sets)	22
1.10.1 Définir des ensembles en Python	22
1.10.2 Accès aux éléments d'un ensemble Python	23
1.10.3 Longueur ou cardinal d'un ensemble Python	23
1.10.4 Opérations : ajouter, supprimer ou mettre à jour un ensemble Python	23
1.10.4.1 Ajouter un ou plusieurs éléments à un ensemble Python	23
1.10.4.2 Supprimer un élément d'un ensemble Python	24
1.10.4.3 Vider un ensemble Python	24
1.10.4.4 Supprimer un ensemble	25
1.10.5 Récapitulatif des méthodes associées à un ensemble Python	25
2 Programmation orientée objet POO en Python	26
2.1 Le concept de POO en Python	26
2.2 Les classes en Python	26
2.3 Les méthodes de classes en Python	27
2.4 Les méthodes statiques	28
2.5 Héritage en Python	28
2.6 Les modules en Python	29
3 Les fichiers en Python	31
3.1 Le module os	31
3.1.1 La méthode os.getlogin()	31
3.1.2 La méthode os.mkdir()	31
3.1.3 La méthode os.getcwd()	31
3.1.4 La méthode os.path	32
3.1.4.1 Tester si un répertoire existe avec la méthode os.path.exists()	32
3.1.4.2 Tester si un chemin est un répertoire ou un fichier avec les méthodes is_dir() et is_file()	32

3.1.4.3 Détermination du chemin absolu à l'aide de la méthode <code>absolute()</code>	32
3.1.4.4 Transformer un chemin en adresse uri avec la méthode <code>as_uri()</code>	33
3.1.4.5 Obtenir le chemin du dossier parent avec la méthode <code>parent</code>	33
3.1.4.6 Récupération du contenu d'un dossier avec la méthode <code>scandir()</code> appliquée à la méthode <code>Path()</code>	33
3.1.4.7 Afficher tous les fichiers d'une extension spécifique via la méthode <code>glob()</code>	34
3.2 Mode d'ouverture d'un fichier	34
3.3 Ouverture et lecture d'un fichier	34
3.3.1 Lecture totale avec la méthode <code>read()</code>	35
3.3.2 Lecture partielle avec la méthode <code>read()</code>	35
3.3.3 Lecture séquentielle caractère par caractère	35
3.3.4 Lecture ligne par ligne avec les méthodes <code>readline()</code> et <code>readlines()</code>	36
3.3.4.1 La méthode <code>readline()</code>	36
3.3.4.2 La méthode <code>readlines()</code>	36
3.3.4.3 Lecture d'un fichier à une position précise avec la méthode <code>readlines()</code>	37
3.4 Lecture et écriture à une position donnée à l'aide de la méthode <code>seek()</code>	37
3.5 Ouverture et écriture dans un fichier en Python	38
3.5.1 Ouverture et écriture dans un fichier existant	38
3.5.2 Créer un nouveau fichier	38
3.6 Récapitulatif des méthodes Python associées à un objet fichier avec description :	39
3.7 Manipulation des fichiers de configuration en Python	39
3.7.1 Le module <code>configparser</code>	39
3.7.2 Format du fichier de configuration	40
3.7.3 Lecture d'un fichier de configuration <code>.ini</code> en Python	40
3.7.4 Mettre à jour un paramètre	41
3.7.5 Lecture des sections d'un fichier de configuration	41
3.7.6 Lire la liste des options avec la méthode <code>options()</code>	42
3.7.7 Lecture totale avec la méthode <code>items()</code>	42
3.7.8 Ajouter une section avec la méthode <code>add_section()</code>	43
3.8 Exercices	44
4 Python et les bases de données	45
4.1 Python et les bases de données SQLite3	45
4.1.1 Création de tables et de bases de données SQLite3	45
4.1.2 Insertion de données	46
4.1.3 Insertion des données de variables dans une table SQLite	46
4.1.4 Affichage des données d'une table SQLite3	47
4.1.4.1 Création d'un cursor pour exécuter une requête de sélection	47
4.1.4.2 Parcourir les résultats de la sélection	47
4.1.5 Éditeur WYSIWYG SQLite3	47
4.2 Python et les bases de données MySQL	48

5 Interfaces graphiques en Python avec Tkinter	49
5.1 Les interfaces graphiques en Python	49
5.2 La bibliothèque graphique Tkinter	49
5.3 Les widgets Tkinter	50
5.4 Exemple de widget Tkinter	51
5.4.1 Le widget Button	51
5.4.2 Le widget label	52
5.4.3 Le champ de saisie Entry	53
5.4.3.1 Définition et syntaxe générale du widget Entry	53
5.4.3.2 Liste des options pour le widget Entry	54
5.4.3.3 Les méthodes associés au widget Entry	54
5.4.4 Associer une action ou évènement à un Widget Tkinter	56
5.4.4.1 Associer une action à un bouton de commande	56
5.4.4.2 Associer un évènement à un bouton de commande (bind action)	56
5.4.4.3 Associer un évènement à un champ Entry (bind action)	56
5.4.5 Le widget Text	56
5.4.5.1 Syntaxe & description du widget Text	56
5.4.5.2 Les options disponibles pour le widget Text	56
5.4.5.3 Les méthodes associées au widget Text	58
5.4.6 Le widget Frame Tkinter	59
5.4.6.1 Syntaxe	59
5.4.6.2 Liste des options d'un widget Frame	60
5.5 Les attributs standard des widgets Tkinter	61
5.6 Les méthodes de gestion des dispositions géométriques des widgets	62
5.6.1 La méthode de disposition géométrique pack()	62
5.6.2 La méthode de disposition géométrique grid()	63
5.6.3 La méthode de disposition géométrique place()	63
5.7 Actions manipulant des widgets Tkinter	64
5.7.1 Action associée à un bouton de commande	64
5.8 Menu Tkinter en Python	64
5.9 Pillow et manipulation d'images sur une fenêtre Tkinter	65
5.9.1 La bibliothèque Pillow	65
5.9.2 Installation de la bibliothèque Pillow	66
5.9.3 Insertion d'image à l'aide de la bibliothèque Pillow	66
5.10 Le module Filedialogue	67
5.10.1 Le module Filedialogue et les méthodes associées	67
5.10.2 La méthode askopenfilename	68
5.10.3 La méthode asksaveasfilename	68
5.10.4 La méthode askdirectory	69
5.11 Le module tkinter.ttk	69
5.11.1 A propos du module tkinter.ttk	69
5.11.2 Usage du module tkinter.ttk	69
5.11.3 Usage de tkinter.ttk Button, Label & Entry	69

5.11.4	Usage de ttk.Combobox	70
5.11.5	Usage de ttk.TreeView	71
5.11.6	Exemple d'utilisation de l'objet Treeview avec une base de données SQLite3	73
5.11.7	Amélioration de l'affichage par une barre de défilement (scrollbar)	75
5.12	Obtenir des informations sur la ligne sélectionné	76
5.13	Mini projet : Création d'un éditeur de texte	77
5.13.1	Création de l'arborescence du système	77
5.13.2	Le fichier library.py	77
5.13.2.1	Importation de la bibliothèque tkinter	77
5.13.2.2	Création de la classe principale du fichier library.py	78
5.13.3	Code finale de la classe library.py	80
5.13.4	Code du fichier principal main.py	82
5.13.5	Apçu de l'éditeur de text	82
6	Le Framework Django	84
6.1	Introduction	84
6.1.1	Qu'est-ce que Django ?	84
6.1.2	Polyvalence	84
6.1.3	Sécurité	84
6.1.4	Architecture	85
6.1.5	Code & maintenance de django	85
6.1.6	Plate formes prises en charge par le framework django	85
6.2	Installation du framework Django	86
6.2.1	Installation Django	86
6.2.2	Vérification de l'installation de Django	86
6.3	Créer votre premier projet Django	87
6.3.1	Création du projet django	87
6.3.2	Demarrage du serveur Django de votre application	88
6.3.3	Création d'une nouvelle application au sein du projet mysite	89
6.3.3.1	Création d'une application nommée TICE_CrmefOujda	89
6.3.3.2	Configuration de l'application TICE_CrmefOujda	90
6.3.4	Exemple de création d'un sondage	91
6.3.4.1	Création du site	91
6.3.4.2	Création du sondage	91

Chapitre 1

Algorithmique en langage Python

Les tutoriels vidéos sont disponibles sur **ma chaine Youtube**

Très Facile :

Très Facile : <https://www.youtube.com/user/InformatiquesFacile>



1.1 Introduction



1.1.1 A propos du langage Python

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.

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

1. **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 a son tour 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 utilisé 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 multiplateforme** : Python peut être exécuté sur tous les principaux systèmes d'exploitations, tels que : Mac OS, Microsoft Windows, Linux 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
 - Education : Python est un excellent langage pour apprendre l'algorithme et la programmation ! Par conséquent largement utilisé aux Lycées, Classes préparatoires, Instituts supérieurs, Universités...

1.2 Installation des outils et premier programme Python

Afin de pouvoir développer en langage Python, vous devez installer les outils nécessaires :

1. Télécharger et installer le langage Python depuis le [site officiel Python](#).
2. 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 rapidité et de sa simplicité d'usage, en plus il est gratuit : [Télcherger WingPersonal](#)

1.3 Les variables et commentaires en Python

1.3.1 Les commentaires en Python

1. Les commentaires sur une seule ligne s'introduisent en insérant le **symbol #** avant le texte.
2. Les commentaires sur plusieurs lignes s'introduisent au sein des triples quotes : `"""`
`....` `"""`

Exemple. Commentaires en python

```
1 # Voici un commentaire sur une seule ligne
2 """ Voici un commentaire
3     sur plusieurs
4     lignes ....
5 """
```

1.3.2 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 de type
3 x = "Hello" # x est maintenant transformé en type string
```

1.3.3 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
```

1.4 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 La structure répétitive For ...

La boucle for, permet d'exécuter des instructions répétés. 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 rang(1,n)** le dernier qui est **n n'est pas inclus!** Cela veut dire que la boucle s'arrête à l'ordre **n-1**.

1.5.3 La stucture 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

```

1.6 Les chaînes de caractères en Python

1.6.1 Définir une chaîne de caractère en Python

comme tous les autres langage, les chaînes de caractères en python sont entourés de guillemets simples ou de guillemets doubles. **"CRMEF OUJDA"** est identique à **'CRMEF OUJDA'**.

Les chaînes peuvent être affichées à l'écran en utilisant la fonction d'impression **print()**.

Comme beaucoup d'autres langages de programmation populaires, les chaînes 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.

Exemple. Obtenez le caractère à la position 1 (rappelez-vous que le premier caractère a la position 0) :

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

1.6.2 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 **sous 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. **counts(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 : 1.6.2)
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 chaine

```
1 s = "CRMEF OUJDA"
2 s1 = s[6:9]
3 print(s1) # affiche OUI
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. méthode **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 examens auront lieu le 15/07/19', ' au CRMEF OUJDA']
2 s = ":".join(l)
3 print(s)
4 # Affiche : Bonjour: les examens 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 examens 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 = ['crme', 'oujda', 'rabat', 'oujda']
3 print(Counter(l))
4 # Affiche : Counter({'oujda': 2, 'crme': 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 les 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.

Méthode Description count () Retourne le nombre de fois qu'une valeur spécifiée apparaît dans un tuple. index () Recherche dans le tuple une valeur spécifiée et renvoie la position de l'endroit où il a été trouvé.

1.9 Les dictionnaires

1.9.1 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
```

1.9.2 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)
```

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

1.9.3.1 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}
```

1.9.3.2 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}
```

1.9.3.3 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,}

```

1.9.3.4 Vider un dictionnaire

Un dictionnaire Python peut être vider 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 : {}

```

1.9.4 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

1.10 Les ensembles Python (Python sets)

1.10.1 Définir des ensembles 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.

1.10.2 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
```

1.10.3 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
```

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

1.10.4.1 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)
```

1.10.4.2 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 **totalemtent aléatoire** !

1.10.4.3 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
```

1.10.4.4 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
```

1.10.5 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

Chapitre 2

Programmation orientée objet POO en Python

2.1 Le concept de POO en Python

La **programmation orientée 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 orientée 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 orientée objet en Python.

2.2 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.

2.3 Les méthodes de classes en Python

Définition 10. Une méthode de classe est une fonction ou procédure nommée au sein de la classe, permettant de définir des propriétés ou comportements des objets d'instances.

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

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

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

2.4 Les méthodes statiques

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

Exemple. methode statique

```

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

2.5 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'âge 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'âge de l'étudiant est : 27

La filière de l'étudiant est : math

2.6 Les modules en Python

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.

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

1. On crée un fichier nommée **myModule.py** et mettons le à **la racine** du répertoire d'installation de Python. Dans notre cas c'est **C :/python/** (il faut que ce chemin soit déclaré sur **la variable d'environnement path**)
2. On introduit un code de quelques fonctions simples sur l fichier **myModule.py** par exemple :

```
1 def somme(x,y) :  
2     return x + y  
3  
4 def division(x,y) :  
5     return x/y
```

3. On crée ensuite un fichier python pour tester le module par exemple **testModule.py** qu'on peut placer sur n'importe quel endroit de notre machine.
4. 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 8 est : ",somme(7,8))
```


Chapitre 3

Les fichiers en Python

3.1 Le module os

Le module os permet de gérer l'arborescence des fichiers, il peut être chargé avec la commande : **import os**

3.1.1 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
```

3.1.2 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 :\
C:\

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

3.1.3 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
```

3.1.4 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.

3.1.4.1 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
```

3.1.4.2 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
```

3.1.4.3 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"

```

3.1.4.4 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

```

3.1.4.5 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 p = 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'

```

3.1.4.6 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'>
```

3.1.4.7 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>

3.2 Mode d'ouverture d'un fichier

La méthode **open()**, permet de créer un objet-fichier 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 crée. 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

3.3 Ouverture et lecture d'un fichier

Pour lire un fichier existant, plusieurs méthode sont disponible :

3.3.1 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
```

3.3.2 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 eecution de la fonction **read(n)** (n = nombre de caractères à lire), le curseur se trouve à la position **n+1**, et donc si on execute la fonction une 2^{ème} fois, la lecture débutera depuis le (n+1)^{ème} caractère.

3.3.3 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 fs.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 = fs.read(1)
5     if c == "":
6         break
7     s = s + c
8 print(s)

```

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

3.3.4.1 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^{ème} exécution, elle pointe vers la n^{ème} ligne.

Exemple. lecture du fichier ligne par ligne

```

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

```

En combinant la méthode `readline()` avec la méthode `while()`, on peut lire la totalité des ligne 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

```

3.3.4.2 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 des 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])
```

3.3.4.3 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[9:21] # extraction depuis le caractère qui se trouve à la position 10
   jusqu'à 20
5 print(result)
```

3.4 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 6ème 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 6ème position
```

3.5 Ouverture et écriture dans un fichier en Python

3.5.1 Ouverture et écriture dans un fichier existant

Pour écrire dans un fichier existant, vous devez ajouter un paramètre 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

Exemple. ouvrir un fichier et y ajouter du contenu :

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

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

```
1 f = open ("myFile.txt", "w")
2 f.write ("Désolé ! J'ai supprimé le contenu!")
3 f.close()
4 # ouvrir et lire le fichier après 'lajout :
5 f = open ("myFile.txt", "r")
6 print (f.read())
```

Remarque 14. la méthode **"w"** écrase tout le fichier.

3.5.2 Créer un nouveau fichier

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

1. **"x" - Create** - crée un fichier, 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

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")
```

3.6 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 chaînes dans le fichier.

3.7 Manipulation des fichiers de configuration en Python

3.7.1 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.

3.7.2 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 host : '127.0.0.1'
4 username : 'root'
5 password : 'root'
```

3.7.3 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'))
```

3.7.4 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'**

3.7.5 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']
```

3.7.6 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']
```

3.7.7 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 premières 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
```

3.7.8 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()
```

3.8 Exercices

Exercice 15. Ecrire un programme Python qui permet de lire un fichier existant sur le bureau nommé **monFichier.txt**. On doit préalablement récupérer le nom d'utilisateur via la commande **os.getlogin()**

Exercice 16. En mode console, tapez python pour accéder à l'interpréteur Python. Tapez ensuite les commandes : **import os** et **from os import chdir**. Créer ensuite un fichier nommé **monFichier.txt** :

```
1 >>> import os
2 >>> f = open("monFichier.txt", 'w')
```

Le fichier **monFichier.txt** a été créé sur quel répertoire ?

Exercice 17. En mode console, en utilisant la commande **chdir()**, changer le répertoire actuel vers le bureau (on doit utiliser la commande **os.getlogin()** pour récupérer le nom d'utilisateur). Créer ensuite un fichier nommé **monFichier.txt** sans préciser le chemin. Ce dernier a été créé sur quel répertoire ?

Exercice 18. En utilisant les méthodes **os.rename()**, créer un programme python permettant de renommer un fichier existant sur le bureau.

Exercice 19. Créer un programme Python permettant de créer un répertoire nommé **monDossier** sur le bureau et de déplacer un fichier qui existe sur le bureau **monFichier.txt** vers le répertoire **monDossier**.

Exercice 20. Créer un programme Python permettant de remplacer les 24 premiers caractères d'un fichier existant sur le bureau par la phrase : **"Le contenu de ce fichier à été modifié"**

Chapitre 4

Python et les bases de données

4.1 Python et les bases de données SQLite3

4.1.1 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

```

4.1.2 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()

```

4.1.3 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()
```

4.1.4 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**...

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

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

4.1.4.2 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**...

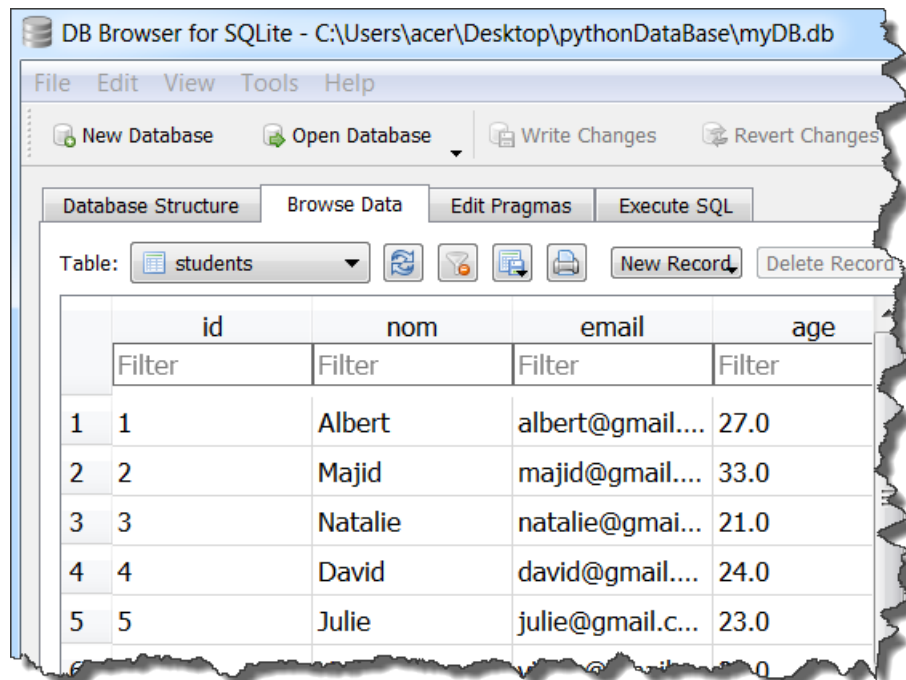
```
1 for row in result :
2     print("ID : ",row[0])
3     print("nom : ",row[1])
4     print("Email : ",row[2])
5     print("Age : ",row[3])
```

4.1.5 Éditeur WYSIWYG SQLite3

Tout a été fait en noir! Jusqu'à présent vous n'avez encore rien vue, ni table ni données...Pour répondre à cette question, je vous informe qu'il y a de nombreux utilitaires per-

mettant 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 donnée que vous venez d'insérer :



4.2 Python et les bases de données MySql

Chapitre 5

Interfaces graphiques en Python avec Tkinter

5.1 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.

5.2 La bibliothèque graphique Tkinter

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

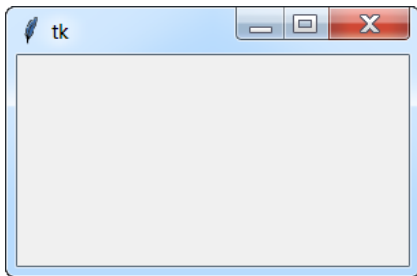
1. **Importez le module Tkinter.**
2. **Créez la fenêtre principale de l'application graphique.**
3. **Ajoutez un ou plusieurs des widgets graphiques.**

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

Ce qui affiche après exécution :



5.3 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 ovals, 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.

5.4 Exemple de widget Tkinter

5.4.1 Le widget Button

Pour créer un **bouton de commande** sur une fenêtre Tkinter, on utilise la syntaxe :

```
1 Nom_du_bouton = Button( Nom_de_la_fenêtre, text = "Texte du bouton", action =  
    action_du_bouton())
```

Exemple. bouton quitter l'application

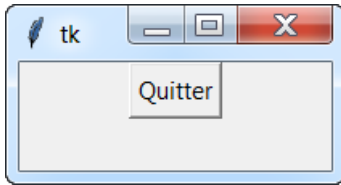
```
1 # -*- coding : utf-8 -*-  
2 from tkinter import*  
3 maFenetre = Tk()  
4  
5 #Création d'un bouton de commande  
6 b = Button(maFenetre , text = "Quitter" , command = quit)  
7
```

```

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"

5.4.2 Le widget label

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")

```

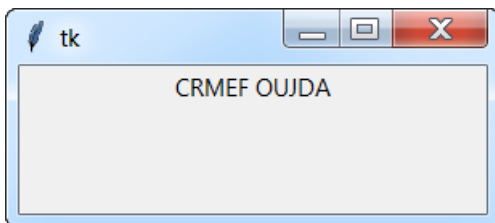
Exemple. label

```

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 21. 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()

```

5.4.3 Le champ de saisie Entry

5.4.3.1 Définition et syntaxe générale du widget Entry

Le champ Entry 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( Nom_de_la_fenêtre,options...)

```

Syntaxe Voici la syntaxe pour créer un widget Entry

```

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

```

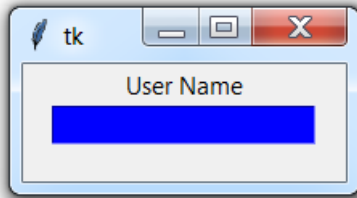
Exemple. Widget Entry

```

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 :



5.4.3.2 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.

5.4.3.3 Les méthodes associés au widget Entry

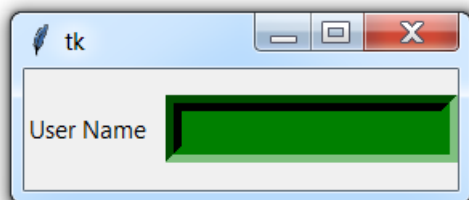
1. **delete** : (first, last = None) : Supprime les caractères du widget, en commençant par celui qui est à l'index first, jusqu'au dernier last

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 :



5.4.4 Associer une action ou évènement à un Widget Tkinter

5.4.4.1 Associer une action à un bouton de commande

5.4.4.2 Associer un évènement à un bouton de commande (bind action)

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

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

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

Exemple. Evènement associé à un champ de saisi :

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

5.4.5 Le widget Text

5.4.5.1 Syntaxe & description du widget Text

Le widget Text offre des fonctionnalités avancées vous permettant d'éditer un texte multilingue 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 pour du widget **Text**

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

Le paramètre parent : représente la fenêtre parent.

options : représente la liste des options pour le widget.

5.4.5.2 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.

5.4.5.3 Les méthodes associées au widget Text

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

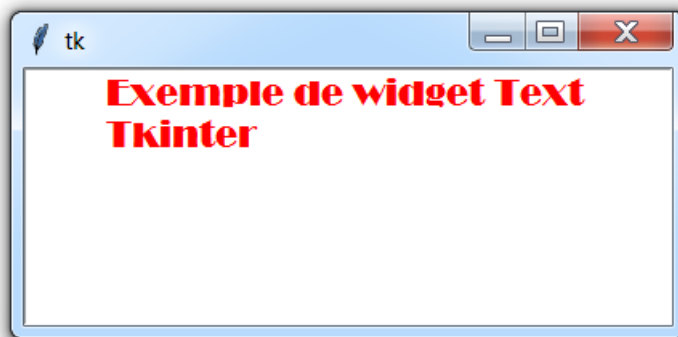
1. **delete (startindex [, endindex])** : cette méthode supprime un caractère spécifique ou une plage de texte.
2. **get (startindex [, endindex])** : cette méthode retourne un caractère spécifique ou une plage de texte.
3. **index (index)** : etourne la valeur absolue d'un index basé sur l'index donné.
4. **insert (index [, string] ...)** : cette méthode insère des chaînes à l'emplacement d'index spécifié.
5. **see(index)** : ette 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 :



5.4.6 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.

5.4.6.1 Syntaxe

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

1. **master** : représente la fenêtre parente.
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.

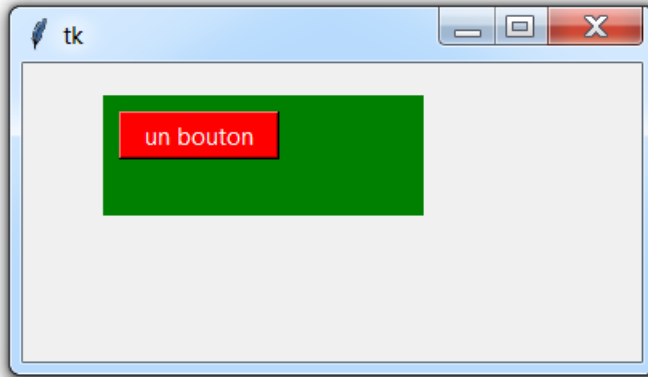
5.4.6.2 Liste des options d'un widget Frame

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** :
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 :



5.5 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 :



5.6 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()**

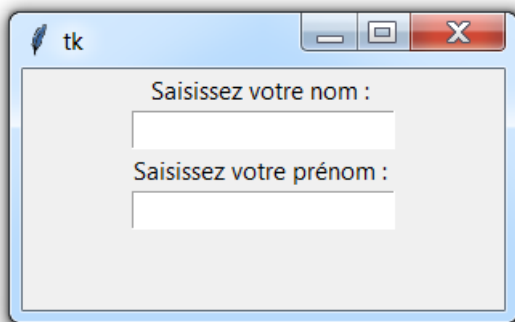
5.6.1 La méthode de disposition géométrique **pack()**

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 :



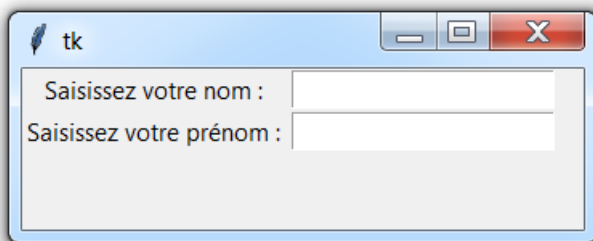
5.6.2 La méthode de disposition géométrique `grid()`

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 :



5.6.3 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()`** :

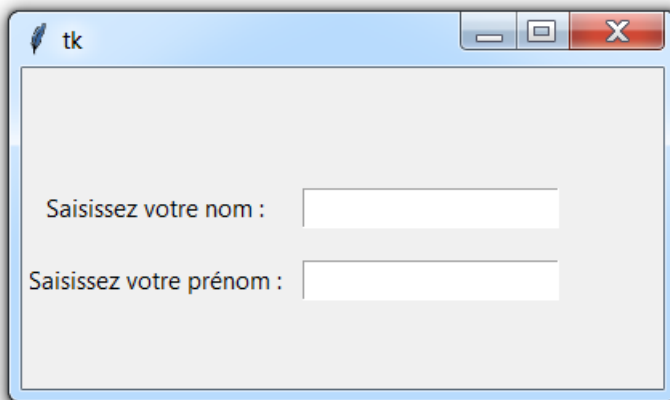
```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3
4 #Création d'une fenêtre Tkinter
5 f = Tk()
6 f.geometry("400x200")
7
8 #Création des widgets
```

```

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 :



5.7 Actions manipulant des widgets Tkinter

5.7.1 Action associée à un bouton de commande

5.8 Menu Tkinter en Python

Le rôle de ce widget est de vous permettre de créer toutes sortes de menus utilisables par vos applications. La création d'un menu se déroule selon les étapes suivantes :

1. Création de la barre des menus

```

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

```

2. Création d'un menu principale :

```

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

```

3. Création d'une commande ou sous menu du menu principal

```

1 menuPrincipal.add_command(label="Sous_menu", command = "..")

```

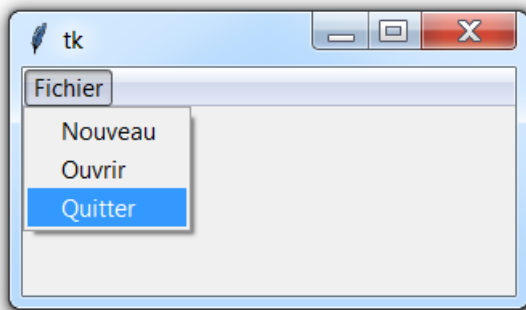
4. Configuration de la barre des menus

```
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()
```

Ce qui affiche après exécution :



5.9 Pillow et manipulation d'images sur une fenêtre Tkinter

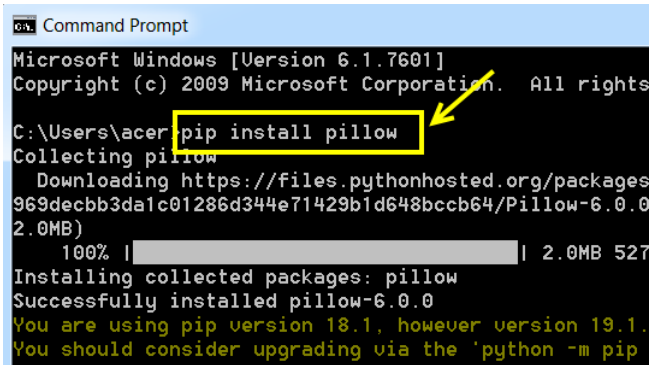
5.9.1 La bibliothèque Pillow

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**.

5.9.2 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/96/9decbb3da1c01286d344e71429b1d648bccb64/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.
  
```

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

Maintenant que la bibliothèque Pillow est installée, 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 :

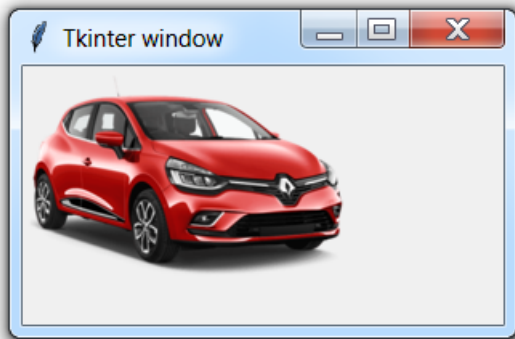
Considérons une image nommée **voiture.png** qui se trouve dans un **dossier images**

Exemple. Insertion d'image

```

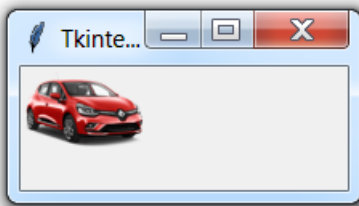
1 from tkinter import *
2 from PIL import Image, ImageTk
3
4 root = Tk()
5 root.title("Tkinter window")
6 root.geometry("300x200")
7
8 # Création de l'objet image
9 load = Image.open("images/voiture.png")
10
11 # Création de la photo à partir de l'objet image
12 photo = ImageTk.PhotoImage(load)
13
14 # 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 22. 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))
```



5.10 Le module Filedialogue

5.10.1 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 **boîte 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**.

5.10.2 La méthode askopenfilename

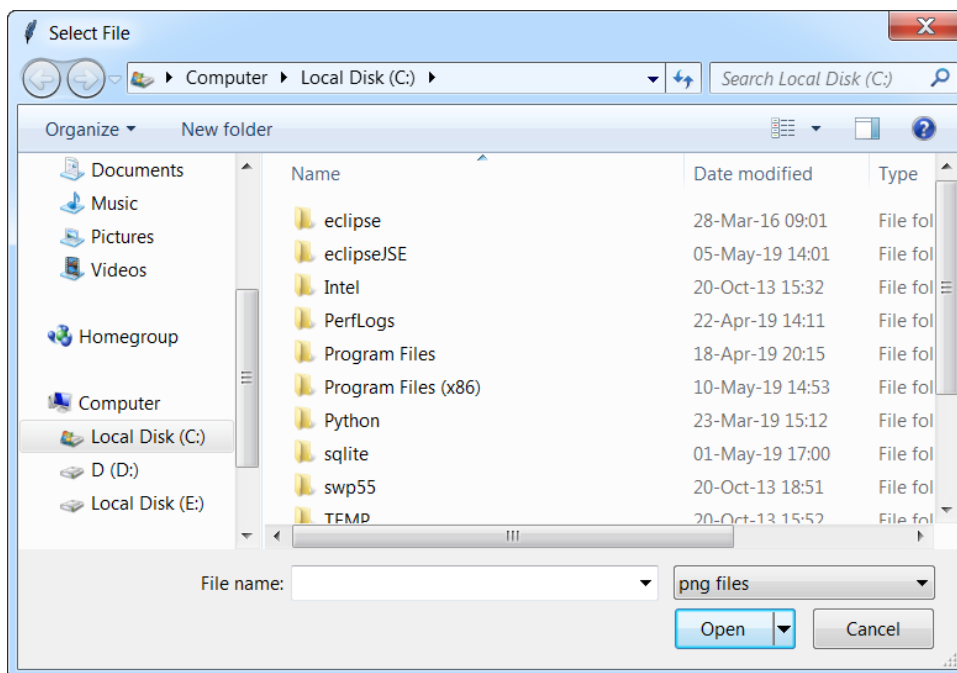
Comme nous l'avons expliqué ci haut, cette méthode affiche une boîte de dialogue qui demande de sélectionner le fichier à ouvrir et récupère le fichier avec son chemin sur une variable. Elle possède les paramètres suivants :

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. méthode 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 :



5.10.3 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 = "/",title = "Select file",filetypes =  
    (("png files","*.png"),("jpeg files","*.jpg"),("all files","*.*"))  
5 print(filename)
```

5.10.4 La méthode askdirectory

Exemple. méthode askdirectory

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

5.11 Le module tkinter.ttk

5.11.1 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 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 les mêmes applications.

5.11.2 Usage du module tkinter.ttk

Afin de pouvoir utiliser le **module tkinter.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 *
```

5.11.3 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(...)
```

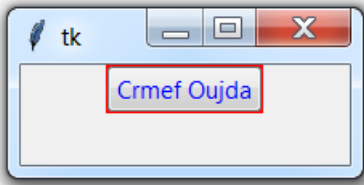
Exemple. Création d'un ttk style pour un bouton

```
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 = "Crmeuf Oujda", style="BW.TButton") b.pack()
8 root.mainloop()
```

Ce qui affiche à l'exécution :



5.11.4 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 23. 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 labelTop.pack()
15
16 # Liste des valeurs d'option de la combobox
17 liste_valeurs=["Acer", "HP", "Dell", "Asus"]
18
19 # Création la liste combobox
20 combo = ttk.Combobox(root, values = liste_valeurs )
21 combo.pack()
22 # Définir l'élément qui s'affiche par défaut
23 combo.current(0)
24 # Associé une bind action à la liste combo
25 combo.bind("<<ComboboxSelected>>", comboAction)
26 root.mainloop()

```

5.11.5 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 donnée 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 importe 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, show = "headings")
2 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", "albert@gmail.com", 27) )
2 tree.insert('', 'end', values = (1, "Majid", "majid@gmail.com", 33) )
3 tree.insert('', 'end', values = (1, "Natalie", "natalie@gmail.com", 21) )
4 #... 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 :

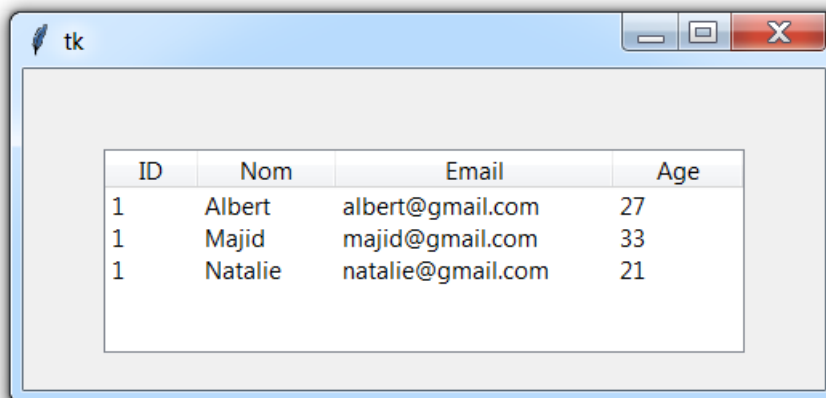
```
1 from tkinter import *
2 from tkinter import ttk
3 root = Tk()
4 root.geometry("500x200")
5
6 # === Création de l'objet Treeview ===
7 tree = ttk.Treeview(root, columns = (1,2,3,4), height = 5, show = "headings")
8 tree.place(x=50,y=50, width=400)
```

```

9
10 # === dimension des colonnes ===
11 tree.column(1, width = 20)
12 tree.column(2, width = 70)
13 tree.column(3, width = 150)
14 tree.column(4,width=50)
15
16 # === Création de l'entête ===
17 tree.heading(1, text="ID")
18 tree.heading(2, text="Nom")
19 tree.heading(3, text="Email")
20 tree.heading(4, text="Age")
21
22 # === Insertion des données ===
23 tree.insert('', 'end', values = (1, "Albert", "albert@gmail.com",27) )
24 tree.insert('', 'end', values = (1, "Majid", "majid@gmail.com",33) )
25 tree.insert('', 'end', values = (1, "Natalie", "natalie@gmail.com",21) )
26
27 root.mainloop()

```

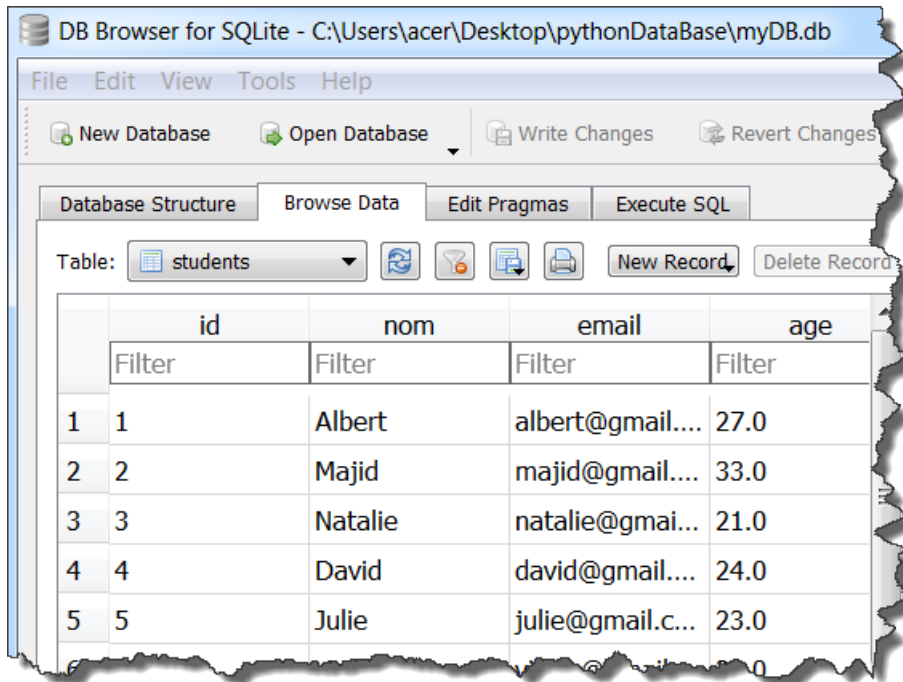
Ce qui affiche après exécution :



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

5.11.6 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, show = "headings")
11 tree.place(x=50,y=50, width=400)
12
13 # === dimension des colonnes ===
14 tree.column(1, width = 20)
15 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 ===

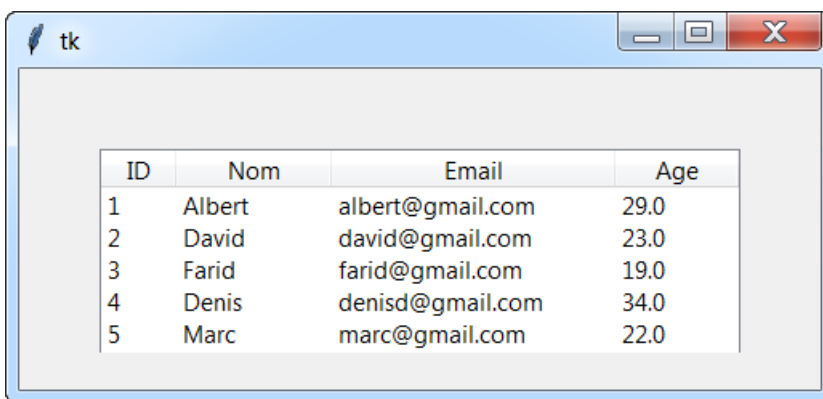
```

```

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[2],row[3]) )
36 conn.close()
37
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

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

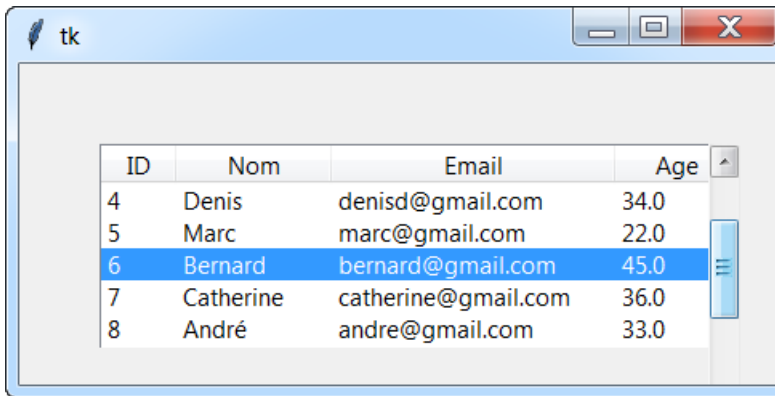
Afin d'améliorer l'affichage des données de l'objet **Treeview**, on peut associer à ce dernier une barre de défilement en ajoutant le code suivant juste après **la ligne n°11** :

```

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 :



ID	Nom	Email	Age
4	Denis	denisd@gmail.com	34.0
5	Marc	marc@gmail.com	22.0
6	Bernard	bernard@gmail.com	45.0
7	Catherine	catherine@gmail.com	36.0
8	André	andre@gmail.com	33.0

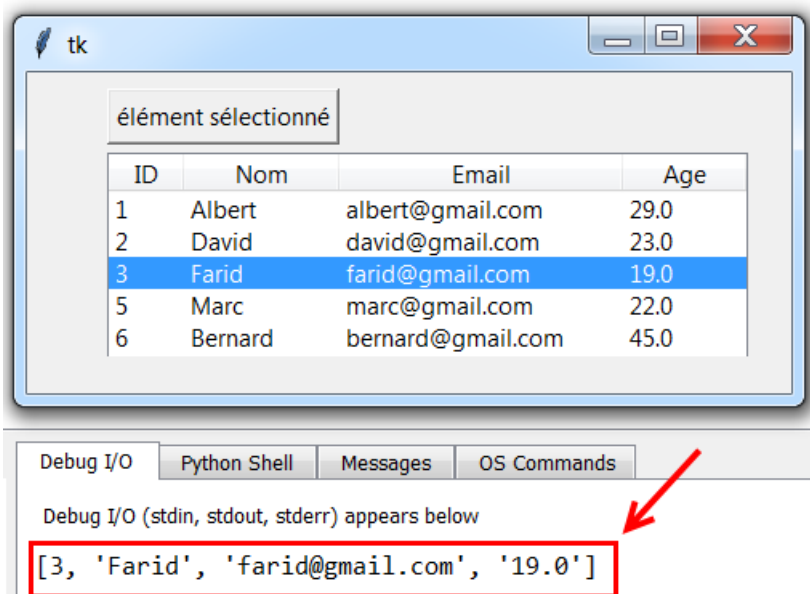
5.12 Obtenir des informations sur la ligne sélectionné

On souhaite parfois obtenir des informations sur une **ligne sélectionnée** de l'objet **Tree-view** 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']
```



élément sélectionné

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
5	Marc	marc@gmail.com	22.0
6	Bernard	bernard@gmail.com	45.0

Debug I/O Python Shell Messages OS Commands

Debug I/O (stdin, stdout, stderr) appears below

```
[3, 'Farid', 'farid@gmail.com', '19.0']
```

Exemple. Supprimer un élément sélectionné

```

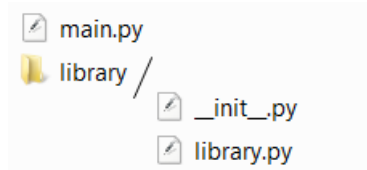
1 def deleteSelectedElement(event) :
2     # Obtention de l'identifiant de l'élément sélectionné
3     id = tree.item(tree.selection())['values'][0]
4     conn = sqlite3.connect('mabase.db')
5     cur = conn.cursor()
6     cur.execute("delete from students where id = {}".format(id))
7     conn.commit()
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, redémarrer l'application

```

5.13 Mini projet : Création d'un éditeur de texte

5.13.1 Création de l'arborescence du système

Avant de commencer à coder, vous devez faire une conception de votre système :



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.
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**

5.13.2 Le fichier library.py

5.13.2.1 Importation de la bibliothèque tkinter

```

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

```

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

5.13.2.2 Création de la classe principale du fichier library.py

```

1 from tkinter import *
2 import os
3 savedFile = {1 : ""}
4 class Win :
5     def __init__(self, master, content) :
6         # Fenêtre principale
7         self.master = master
8         # Main Text Widget
9         self.content=content
10    # Création de la fenêtre tkinter
11    def create(self) :
12        self.master = Tk()
13        self.master.title("Editeur de Texte")
14        self.master.geometry("700x550")
15    # Méthode qui ajoute la zone de texte
16    def add_text(self) :
17        self.content = Text(self.master)
18        self.content.pack(expand=1, fill='both')
19    # Génération de la fenêtre principale
20    def generate(self) :
21        self.master.mainloop()

```

Ajouter ensuite une méthode **add_menu()** qui permet d'ajouter le menu à la fenêtre

```

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       self.master.config(menu = menuBar)
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")

```


A chaque menu est associée une commande : **command = nom_de_la_commande**. Vous devez donc créer une action pour chaque menu :

```

1 #=====
2 #  - Définition des actions des menus
3 #=====
4 #-----#
5 - actions du menu Fichier
6 #-----
7 def quitter(self) :
8     self.master.quit()
9 def nouveau(self) :
10     os.popen("python main.py")
11 def fopen(self) :
12     file = self.master.filename = filedialog.askopenfilename(initialdir = "/",title = "
    Select File",filetypes = (("Text Files","*.txt"),("all files","*.*")))
13     fp = open(file,"r")
14     r = fp.read()
15     self.content.insert(1.0,r)
16     # Menu Enregistrer sous
17 def saveAs(self) :
18     # create save dialog
19     fichier=self.master.filename=filedialog.asksaveasfilename(initialdir = "/",title = "
    Enregistrer Sous\
    fichiers","*.*"))
20     fichier = fichier + ".txt"
21     # Utilisation du dictionnaire pour stocker le fichier
22     savedFile[1] = fichier
23     f = open(fichier,"w")
24     s = self.content.get("1.0",END)
25     f.write(s)
26     f.close()
27     # menu Enregistrer
28 def save(self) :
29     if(savedFile[1] == "") :
30         self.saveAs()
31     else :
32         f = open(savedFile[1],"w")
33         s = self.content.get("1.0",END)
34         f.write(s)
35         f.close()
36 #-----
37 #  - actions du menu Edition
38 #-----
39 def copy(self) :
40     self.content.clipboard_clear()
41     self.content.clipboard_append(self.content.selection_get())
42     def past(self) :
43         self.content.insert(INSERT, self.content.clipboard_get())
44 def cut(self) :
45     self.copy()
46     self.content.delete("sel.first","sel.last")

```

Remarque 24. Les actions associées aux menus doivent être placées avant les fonctions qui définissent les menus

5.13.3 Code final de la classe library.py

Code sur pastbin : <https://pastebin.com/dHusqwyT>

```

1  # -*- coding : utf-8 -*-
2  #
3  # Author      : Y. DERFOUFI
4  # Compagny    : CRMEF OUJDA
5  #
6  from tkinter import *
7  import os
8  savedFile = {1 : ""}
9
10 #=====
11 # 1 – Classe de la fenêtre principale
12 #=====
13 class Win :
14     def __init__(self, master, content) :
15         # Fenêtre principale
16         self.master = master
17         # Main Text Widget
18         self.content = content
19         # Création de la fenêtre tkinter
20     def create(self) :
21         self.master = Tk()
22         self.master.title("Editeur de Texte")
23         self.master.geometry("700x550")
24         # Méthode qui ajoute la zone de texte
25     def add_text(self) :
26         self.content = Text(self.master)
27         self.content.pack(expand=1, fill='both')
28         # Génération de la fenêtre principale
29     def generate(self) :
30         self.master.mainloop()
31
32 #=====
33 # 2 – Définition des actions des menus
34 #=====
35 #
36 # 2.1 – actions du menu Fichier
37 #
38     def quitter(self) :
39         self.master.quit()
40     def nouveau(self) :
41         os.popen("python main.py")
42     def fopen(self) :
43         file = self.master.filename = filedialog.askopenfilename(initialdir = "/", title = "
44         Select File", filetypes = (("Text Files", "*.txt"), ("all files", "*.*")))
45         fp = open(file, "r")
46         r = fp.read()
47         self.content.insert(1.0, r)
48         # Menu Enregistrer sous
49     def saveAs(self) :
50         # create save dialog
51         fichier = self.master.filename = filedialog.asksaveasfilename(initialdir = "/", title =

```

```

    "Enregistrer Sous\                ", filetypes = (("Fichier Texte", "*.txt"), ("Tous les
    fichiers", "*.*")))
51     fichier = fichier + ".txt"
52     # Utilisation du dictionnaire pour stocker le fichier
53     savedFile[1] = fichier
54     f = open(fichier, "w")
55     s = self.content.get("1.0", END)
56     f.write(s)
57     f.close()
58     # menu Enregistrer
59     def save(self):
60         if(savedFile[1] == ""):
61             self.saveAs()
62         else:
63             f = open(savedFile[1], "w")
64             s = self.content.get("1.0", END)
65             f.write(s)
66             f.close()
67     #
68     # 2.2 – actions du menu Edition
69     #
70     def copy(self):
71         self.content.clipboard_clear()
72         self.content.clipboard_append(self.content.selection_get())
73     def past(self):
74         self.content.insert(INSERT, self.content.clipboard_get())
75     def cut(self):
76         self.copy()
77         self.content.delete("sel.first", "sel.last")
78
79     #=====
80     # 2 – Méthodes d'ajout des menus
81     #=====
82     def add_menu(self):
83         # 1 – Création de la barre des menus
84         menuBar = Menu(self.master)
85         # 2 – Création du menu Fichier
86         menuFichier = Menu(menuBar, tearoff=0)
87         menuBar.add_cascade(label = "Fichier", menu=menuFichier)
88         # Création des sous menus du menu Fichier
89         menuFichier.add_command(label="Nouveau", command=self.nouveau)
90         menuFichier.add_command(label="Ouvrir", command=self.fopen)
91         menuFichier.add_command(label="Enregistrer", command=self.save)
92         menuFichier.add_command(label="Enregistrer sous", command=self.saveAs)
93         menuFichier.add_command(label="Quitter", command = self.quitter)
94         self.master.config(menu = menuBar)
95         #3 – Création du Menu Edition
96         menuEdition= Menu(menuBar, tearoff=0)
97         menuBar.add_cascade(label = "Edition ", menu=menuEdition)
98         menuEdition.add_command(label="Annuler")
99         menuEdition.add_command(label="Rétablir")
100        menuEdition.add_command(label="Copier", command=self.copy)
101        menuEdition.add_command(label="Couper", command = self.cut)
102        menuEdition.add_command(label="Coller", command=self.past)
103        #4 – Création du Menu Outils

```

```

104     menuOutils = Menu(menuBar, tearoff=0)
105     menuBar.add_cascade(label = "Outils", menu = menuOutils)
106     menuOutils.add_command(label="Préférences")
107     # Création du Menu Aide
108     menuAide = Menu(menuBar, tearoff=0)
109     menuBar.add_cascade(label = "Aide", menu = menuAide)
110     menuAide.add_command(label="A propos")

```

5.13.4 Code du fichier principal main.py

Code sur **pastbin** : <https://pastebin.com/wxAp9wej>

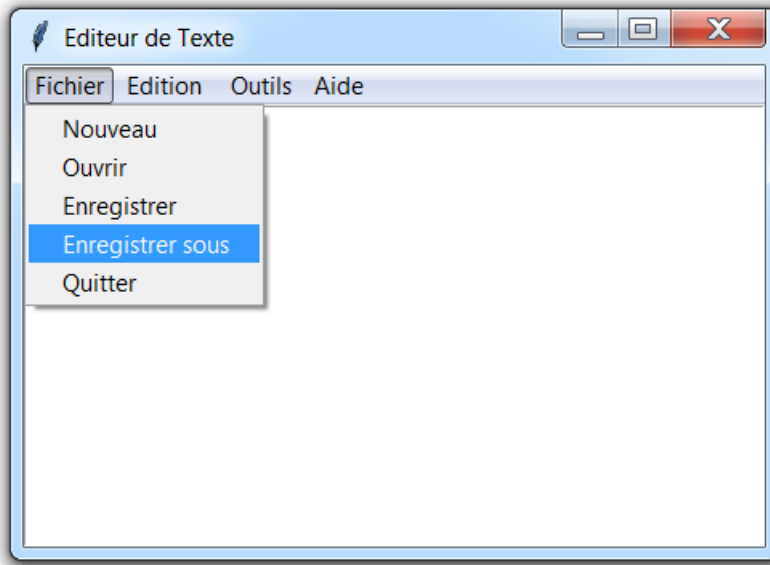
```

1  # -*- coding : utf-8 -*-
2  # Importation des bibliothèques nécessaires
3  from tkinter import *
4  from tkinter import filedialog
5
6  # Importation du module library.py
7  from library.library import *
8
9  # Création du dictionnaire qui stock les objets files
10 savedFile = {1 : ""}
11
12 # Création d'une instance sur la classe principale
13 root = Win("root", "c")
14
15 # Appel des méthodes qui créent l'objet fenêtre avec tous ces composants
16 root.create()
17 root.add_text()
18 root.add_menu()
19 root.generate()

```

5.13.5 Aperçu de l'éditeur de text

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



Chapitre 6

Le Framework Django

6.1 Introduction

6.1.1 Qu'est-ce que Django ?

Django est un framework(c.a.d : une infrastructure de développement ou cadriceil son principal but et de rendre le développement facile et accéléré. Le terme framework est un mot anglicisme composé de frame et work et qui veut dir un cadre de travail son rôle est de rendre le développement web rapide et efficace). Django est un framework ataché au langage Python de haut niveau qui permet le développement rapide et performant de sites Web sécurisés et maintenables. – Construit par des développeurs expérimentés, Django prend en charge la plupart des problèmes liés au développement Web. Vous pouvez ainsi vous concentrer sur le développement de votre application sans avoir à réinventer la roue, c.a.d sans vous casser la têtes avec les codes du lagage. Django est une source libre et ouverte, une communauté dynamique et active, une excellente documentation et de nombreuses options pour une assistance complètement gratuite.

6.1.2 Polyvalence

Django peut être (et a été déjà) utilisé pour créer presque tous les types de sites Web, des systèmes de gestion de contenu, les forums les wikis... en passant par les réseaux sociaux et les sites d'informations. Il peut fonctionner avec n'importe quelle infrastructure côté client et peut fournir du contenu dans presque tous les formats (y compris HTML, flux RSS, JSON, XML, etc.).

Il faut noter qu'aujourd'hui django est utilisé par de nombreux grand sites comme 'Instagram, Mozilla, Google, Pinterest, et même la NASA !

6.1.3 Sécurité

Django aide les développeurs à éviter de nombreuses erreurs de sécurité courantes en fournissant un cadre conçu pour « faire le nécessaire » afin de protéger automatiquement le site Web. Par exemple, Django offre un moyen sécurisé de gérer les comptes d'utilisateurs et

les mots de passe, en évitant les erreurs courantes telles que le stockage d'informations de session dans des cookies vulnérables (les cookies ne contiennent qu'une clé et les données sont stockées dans la base de données) ou stockent directement des mots de passe. plutôt qu'un mot de passe hash.

Un hachage de mot de passe est une valeur de longueur fixe créée en envoyant le mot de passe via une fonction de hachage cryptographique. Django peut vérifier si un mot de passe entré est correct en l'exécutant via la fonction de hachage et en comparant la sortie à la valeur de hachage stockée. Cependant, en raison de la nature « à sens unique » de la fonction, même si une valeur de hachage stockée est compromise, il est difficile pour un attaquant de trouver le mot de passe d'origine.

Django active la protection contre de nombreuses vulnérabilités par défaut, notamment l'injection SQL, les scripts intersites, la falsification de requêtes intersites et le détournement de clics (voir Sécurité du site Web pour plus de détails sur ces attaques). Évolutif.

6.1.4 Architecture

Django utilise une architecture «sans partage» basée sur des composants (chaque partie de l'architecture est indépendante des autres et peut donc être remplacée ou modifiée si nécessaire). Disposer d'une séparation claire entre les différentes parties signifie qu'il peut évoluer en augmentant le trafic en ajoutant du matériel à tout niveau : serveurs de mise en cache, serveurs de base de données ou serveurs d'applications. Certains des sites les plus actifs ont réussi à adapter Django à leurs besoins (par exemple, Instagram et Disqus, pour ne nommer que deux).

6.1.5 Code & maintenance de django

Le code Django est écrit en utilisant des principes et des modèles de conception qui encouragent la création de code maintenable et réutilisable. En particulier, il utilise le principe «Ne pas répéter» (DRY) pour qu'il n'y ait pas de duplication inutile, ce qui réduit la quantité de code. Django encourage également le regroupement des fonctionnalités associées dans des « applications » réutilisables et, à un niveau inférieur, le code associé dans des modules (suivant le modèle MVC (Model View Controller)). Portable.

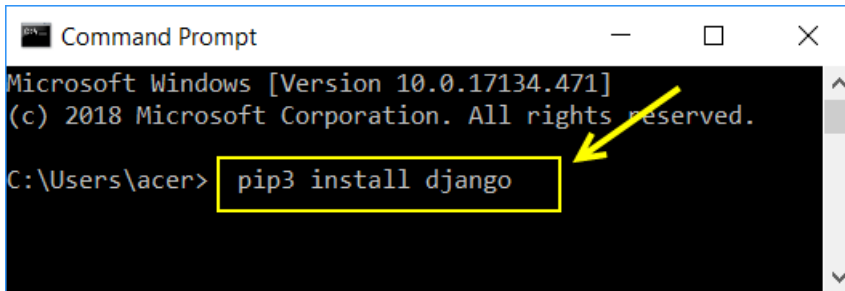
6.1.6 Plate formes prises en charge par le framework django

Django est écrit en Python, qui tourne sur de nombreuses plateformes. Cela signifie que vous n'êtes pas lié à une plate-forme de serveur particulière et que vous pouvez exécuter vos applications sur de nombreux types de Linux, Windows et Mac OS X. De plus, Django est bien pris en charge par de nombreux fournisseurs d'hébergement Web, qui fournissent souvent des infrastructures et des environnements spécifiques. documentation pour l'hébergement de sites Django.

6.2 Installation du framework Django

6.2.1 Installation Django

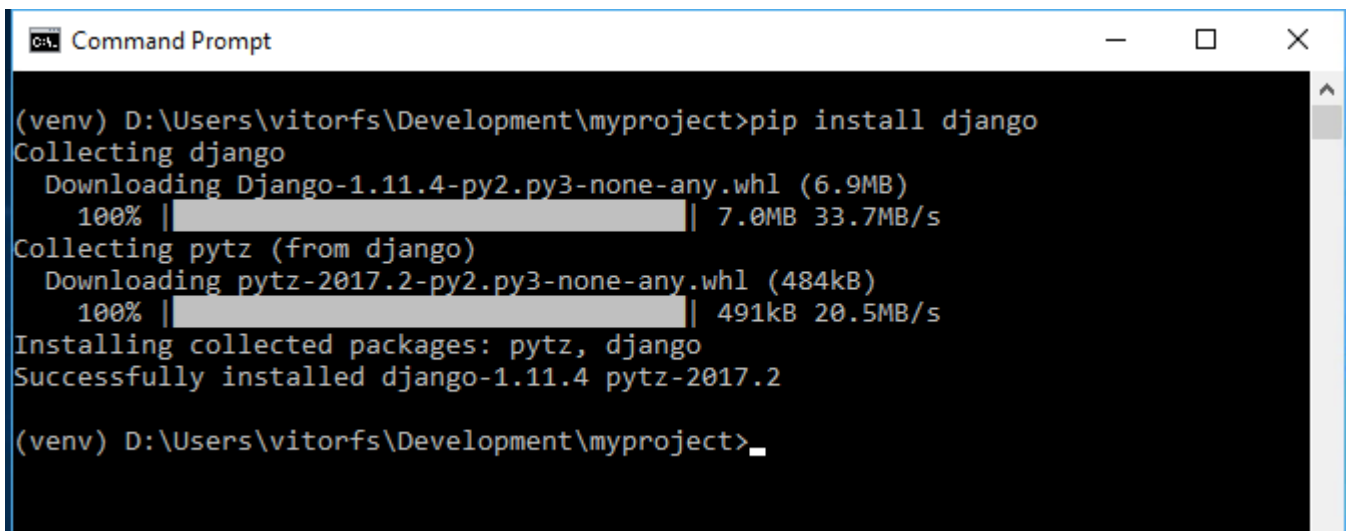
Maintenant que Python est installé et que nous exécutons un environnement virtuel, l'installation de Django est une opération très facile, il vous suffit de taper la commande suivante dans votre invite de commande cmd :



```
Command Prompt
Microsoft Windows [Version 10.0.17134.471]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\acer> pip3 install django
```

Ceci indiquera à **pip** d'installer la dernière version de **Django 2.1** dans votre environnement virtuel. La sortie de votre commande devrait ressembler à ceci :



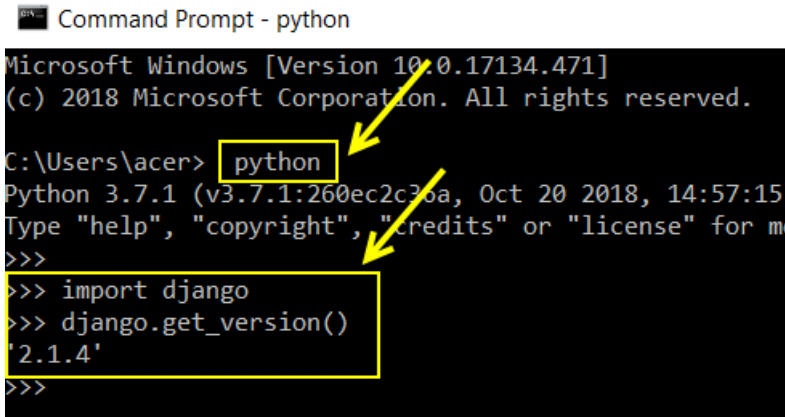
```
Command Prompt

(venv) D:\Users\vitorfs\Development\myproject>pip install django
Collecting django
  Downloading Django-1.11.4-py2.py3-none-any.whl (6.9MB)
    100% |#####| 7.0MB 33.7MB/s
Collecting pytz (from django)
  Downloading pytz-2017.2-py2.py3-none-any.whl (484kB)
    100% |#####| 491kB 20.5MB/s
Installing collected packages: pytz, django
Successfully installed django-1.11.4 pytz-2017.2

(venv) D:\Users\vitorfs\Development\myproject>_
```

6.2.2 Vérification de l'installation de Django

Pour tester l'installation, démarrez l'interpréteur interactif Python depuis l'invite de commande de votre environnement en tapant **python** et en appuyant sur la touche Entrée du clavier. Si l'installation a réussi, vous devriez pouvoir importer le module django et taper la commande **django.get_version()** :



```
Command Prompt - python
Microsoft Windows [Version 10.0.17134.471]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\acer> python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15)
Type "help", "copyright", "credits" or "license()" for more
>>>
>>> import django
>>> django.get_version()
'2.1.4'
>>>
```

Dans notre cas la version **2.1.4** de **django** est installé !

6.3 Créer votre premier projet Django

6.3.1 Création du projet django

Si vous utilisez Django pour la première fois, vous devrez prendre en compte de la configuration initiale. En particulier, vous devrez générer automatiquement du code qui établit un **projet Django, c.a.d** un ensemble de paramètres pour une instance de **Django**, y compris la configuration de la base de données, des options spécifiques à Django et des paramètres spécifiques à une application.

A partir de la ligne de commande, accédez au répertoire dans lequel vous souhaitez stocker votre code, puis exécutez la **commande suivante** :

```
1 django-admin startproject mysite
```

Cela créera un répertoire **mysite** dans votre répertoire actuel. Voici un aperçu du contenu de votre site que vous venez de créer avec la commande **django-admin startproject** :

```
mysite/
  manage.py
  mysite/
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

1. **Le répertoire racine mysite/** : designe le conteneur de votre projet. Il englobe tout le contenu de votre site.
2. **manage.py** : contient des codes qui vous permet d'interagir avec votre projet Django via l'utilitaire de ligne de commande.

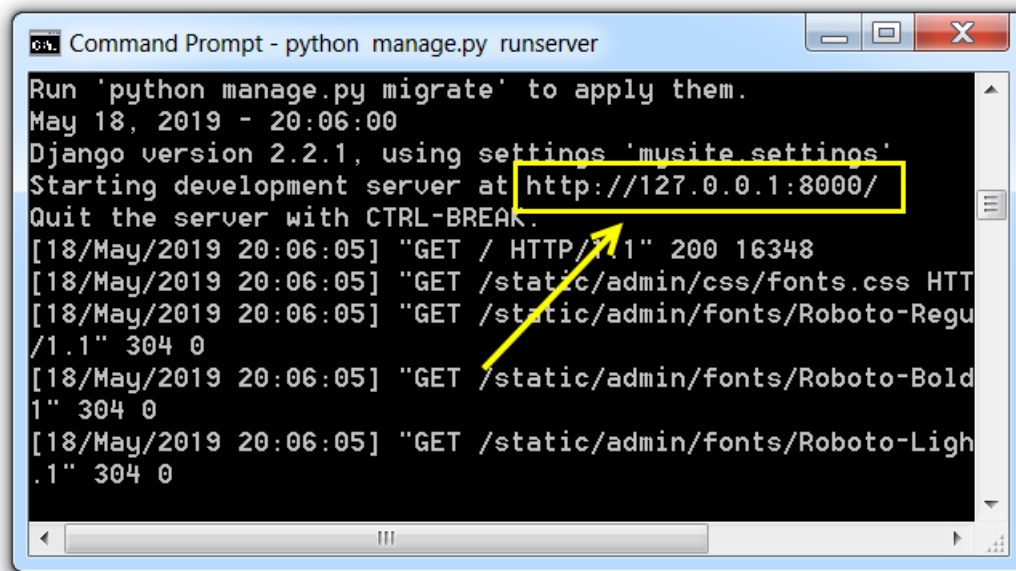
3. **Le répertoire interne `mysite/`** : est le **package Python** actuel de votre projet. Son nom est le nom du package Python que vous devrez utiliser pour importer quoi que ce soit à l'intérieur de celui-ci.
4. **`mysite/___init___`.py** : fichier vide indiquant à Python que ce répertoire doit être considéré comme un **package Python**.
5. **`mysite/settings`.py** : contient les paramètres de configuration pour votre projet Django.
6. **`mysite/urls`.py** : contient les adresses url de votre projet. Chaque nouvelle url doit être déclarée ici dans ce fichier.
7. **`mysite/wsgi`.py** : Un point d'entrée pour les serveurs Web compatibles **WSGI** nécessaire à la mise en oeuvre de votre projet.

6.3.2 Demarrage du serveur Django de votre application

Pour vérifiez que votre projet Django fonctionne correctement, allez dans le répertoire externe `mysite` de votre application, et exécutez la suivante :

```
1 python manage.py runserver
```

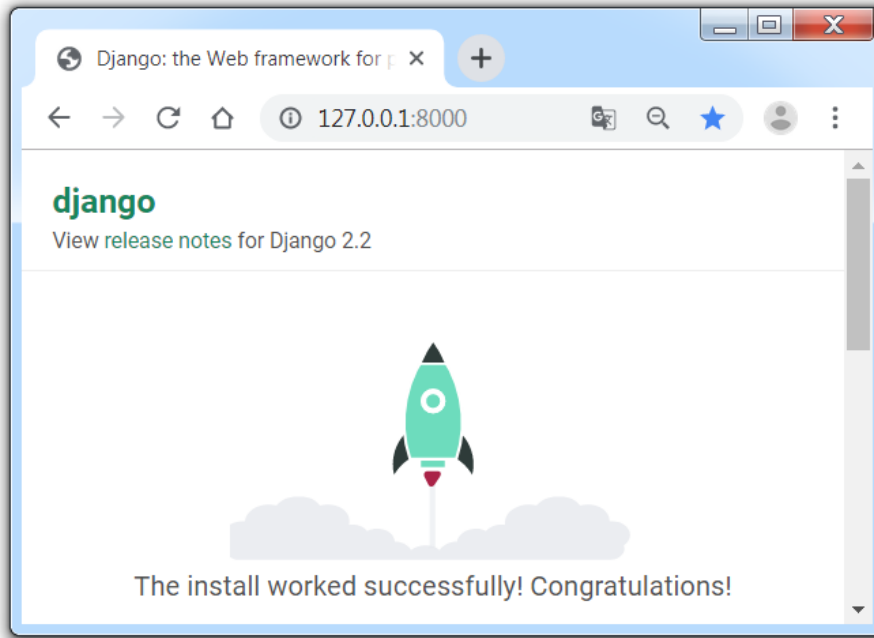
Vous verrez la sortie suivante sur la ligne de commande :



```
Command Prompt - python manage.py runserver
Run 'python manage.py migrate' to apply them.
May 18, 2019 - 20:06:00
Django version 2.2.1, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[18/May/2019 20:06:05] "GET / HTTP/1.1" 200 16348
[18/May/2019 20:06:05] "GET /static/admin/css/fonts.css HTTP/1.1" 304 0
[18/May/2019 20:06:05] "GET /static/admin/fonts/Roboto-Regular.woff2 HTTP/1.1" 304 0
[18/May/2019 20:06:05] "GET /static/admin/fonts/Roboto-Bold.woff2 HTTP/1.1" 304 0
[18/May/2019 20:06:05] "GET /static/admin/fonts/Roboto-Light.woff2 HTTP/1.1" 304 0
```

Qui vous indique que le serveur est démarré à l'adresse **[http ://127.0.0.1 :8000](http://127.0.0.1:8000/)**

Si vous souhaitez accéder à la home page de votre serveur, il suffit de taper l'adresse indiquée ci-dessus sur la barre d'adresse de votre navigateur :



Remarque 25. Le serveur choisi le port **8000** par défaut ! Mais il arrive parfois que ce port soit occupé par d'autres processus, et que vous devriez le changer, il suffit alors pour cela d'ajouter le numéro de port de votre choix juste après la commande **runserver** :

Exemple. changement du port

```
1 python manage.py runserver 5050
```

Dans ce cas le serveur sera accessible à l'adresse : <http://127.0.0.1:5050>

Remarque 26. Pour arrêter le serveur, il suffit de taper la combinaison de touche : **Ctrl + C**

6.3.3 Création d'une nouvelle application au sein du projet mysite

6.3.3.1 Création d'une application nommée TICE_CrmefOujda

Un projet de site web est généralement constitué d'un ensemble d'applications. A cet effet nous allons créer une nouvelle application nommé **TICE_CrmefOujda** au sein de notre projet **mysite**. Pour ce faire, accédez à l'aide de l'invite de **commande cmd** au répertoire de votre site **mysite** et tapez la commande :

```
1 python manage.py startapp TICE_CrmefOujda
```

Vous obtenez alors, un répertoire nommé **TICE_CrmefOujda** au sein de votre projet **mysite** :

```
TICE_CrmefOujda /
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

6.3.3.2 Configuration de l'application TICE_CrmefOujda

Et maintenant afin que la nouvelle application soit opérationnelle, il faudra préalablement effectuer quelques configurations :

1. Créer un fichier **TICE_CrmefOujda/urls.py** et introduisez au sein duquel le code suivant :

```
1 from django.urls import path
2 from . import views
3 urlpatterns = [
4     path( '', views.index, name='index' ),
5 ]
```

Ce code importe le contenu du fichier **view.py**

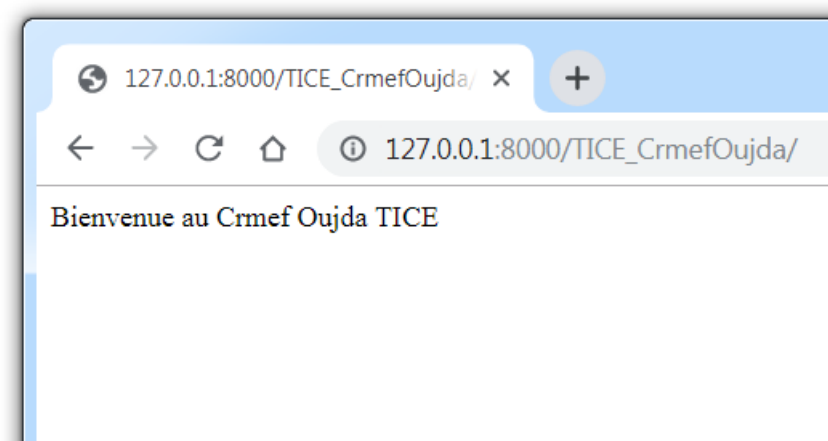
2. Créer le contenu du fichier **TICE_CrmefOujda/view.py** :

```
1 from django.http import HttpResponse
2 def index(request) :
3     return HttpResponse("Bienvenue au Crmef Oujda TICE")
```

3. La dernière étape consiste à faire pointer la configuration d'URL racine vers le module **TICE_CrmefOujda.urls**. Dans **mysite/urls.py**, ajoutez une importation `django.urls.include` et insérez un appel **include()** dans la liste `urlpatterns`, ce qui donnera :

```
1 from django.contrib import admin
2 from django.urls import include, path
3 urlpatterns = [
4     path( 'TICE_CrmefOujda/' , include( 'TICE_CrmefOujda.urls' )),
5     path( 'admin/' , admin.site.urls), ]
```

4. Demarrer maintenant votre serveur à l'aide de la commande : **python manage.py runserver** et tapez l'adresse url : **http ://127.0.0.1 :8000/TICE_CrmefOujda/** dans votre navigateur pour voir la page d'accueil de votre nouvelle application :



6.3.4 Exemple de création d'un sondage

6.3.4.1 Création du site

```
1 python manage.py startproject monsite
```

6.3.4.2 Création du sondage

```
1 cd monsite
2 python manage.py startapp sondage
```

Références

-
- [1] Documentation officielle Python : <https://docs.python.org/fr/3/>
 - [2] Gérard Swinnen. Apprendre à programmer avec Python 3. Eyrolles
 - [3] Magnus Lie Hetland. Beginning Python From Novice to Professional, Second Edition. ISBN-13 (pbk) : 978-1-59059-982-2. Copyright 2008.
 - [4] Mark Lutz. learning Python . ISBN : 978-1-449-35573-9. 5 ème édition.
 - [5] Burkhard A. Meier. Python GUI Programming Cookbook. Copyright © 2015 Packt Publishing. ISBN 978-1- 78528-375-8.
 - [6] Bhaskar Chaudhary. Tkinter GUI Application Development Blueprints. Copyright © 2015 Packt Publishing. ISBN 978-1-78588-973-8