Programmation en langage Python

Younes Derfoufi

Programmation en langage Python

Younes. Derfoufi Enseignant au CRMEF OUJDA

11 mai 2019

Table des matières

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 chevronés.

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 lanagage Python : Pycharm, Wing, PyScripter, Spyder...
3. **Python est orienté objet :** la puissance du langage python est fortement marquée par son aspet 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 a connu dans ce contexte un grand essor : des **dizaines de milliers de bibliothèques** sont disponibles à l'aide de l'**outil pip** de **gestion des packages**.
4. **Simplicité et lisibilité du code :** Python a une syntaxe simple qui le rend approprié pour apprendre la programmation en tant que premier langage. L'apprentissage est plus fluide et rapide que d'autres langages tels que **Java**, qui nécessite très tôt une connaissance de la programmation orientée objet ou du **C/C++** qui nécessite de comprendre les pointeurs. Néanmoins, il est possible d'en apprendre davantage sur la programmation orientée objet en Python lorsqu'il est temps. Par conséquent, Python peut être utilisé comme prototype et peut être implémenté dans un autre langage de programmation après avoir testé le code.
5. **Python est open source donc gratuit :** Python étant un langage de programmation open source, il est gratuit et permet une utilisation illimitée. Avec cette licence open-source, il peut être modifié, redistribué et utilisée commercialement. Avec cette licence, Python est devenu robuste, doté de capacités évolutives et portables et est devenu un langage de programmation largement utilisé.
6. **Python est multiplateforme :** Python peut être exécuté sur tous les principaux systèmes d'exploitations, tels que : Mac OS, Microsoft Windows, Linus et Unix... Ce langage de programmation offre une meilleure expérience de travail avec n'importe quel système d'exploitation.
7. **Python est très puissant en terme de production :** la puissance du langage Python a été démontré sur le terrain du développement :

— Développement Web, en utilisant les frameworks Django, Flask, Pylons
— Science des données et visualisation à l'aide de Numpy, Pandas et Matplotlib
— Applications de bureau avec Tkinter, PyQt, Gtk, wxWidgets et bien d'autres..
— Applications mobiles utilisant Kivy ou BeeWare
— Education : Python est un excellent langage pour apprendre l'algorithmique 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 insérrant le **symbol #** avant le texte.
2. Les commentaires sur plusieurs lignes s'introduisent au sein des triples quotes : """ ...."""

Exemple. Commentaires en python

```
# Voici un commentaire sur une seule ligne
""" Voici un commentaire
    sur plusieurs
    lignes ....
"""
```

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

```
x = 7
y = "Albert"
print(x)
print(y)
```

Une variable python possède toujours un type, même s'il est non déclarée. le type se définie 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.

```
x = 3
# x est de type
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 prédefinies 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 :

```
if(condition):
    instructions...
else:
    autres instructions...
```

Exemple. structure if ... else...

```
# -*- coding: utf-8 -*-
age = 19
if(age >= 18):
    print("Vous êtes majeur !")
else:
    print("Vous êtes mineur !")
# affiche vous êtes majeur
```

1.5.2 La structure répétitive For ...

La boucle for, permet d'exécuter des instructions répétés. Sa sytaxe est :

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

Exemple. affichage des 10 premiers nombres

```
# -*- coding: utf-8 -*-
for i in range(1,11):
    print(i)
#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 sytaxe est :

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

Exemple. affichage des 10 premiers entiers avec la boucle while

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

1.6 Les chaines de caractères en Python

1.6.1 Définir une chaine de caractère en Python

comme tous les autres langage, les chaines 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 chaines de caractères en Python

Le langage Python est doté d'un grand nombre de fonctions permettant la manipulation des chaines de caractères : calcul de la **longueur de la chaine,** transformation en **majuscule** et **minuscule**, extraire une **sosus chaine**...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 chaine de s depuis la ième position jusqu'à la jème non incluse
43. **s[i :] :** Extrait la sous chaine de s depuis la ième position jusqu'à la fin de la chaine
44. **s[: j] :** Extrait la sous chaine de s depuis le début jusqu'à la jème position non incluse

Exemple. transformation d'une chaine 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 chaine

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

Exemple. String.format

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

Exemple. extraire une sous chaine

```
s = "CRMEF OUJDA"
s1 = s[6 :9]
print(s1)    # affiche OUJ
s2 = s[6 :]
print(s2)    # affiche OUJDA
s3 = s[ :4]
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. —

```
#Création d'une liste
myList = ["Python", "Java", "PHP"]
# Affichage de la liste
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 :

```
myList = ["Python", "Java", "PHP"]
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 :

```
myList = ["Python", "Java", "PHP"]
myList[1]="Oracle"
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 :

```
myList = ["Formation", "Python", "au CRMEF OUJDA"]
for x in myList:
   # Afficher tous les éléments de la liste un par un
    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 :

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

```
myList = ["Formation", "Python", "au CRMEF"]
myList.append ("OUJDA")
print (myList)
#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 :

```
myList = ["Python", "Java", "PHP"]
myList.insert (1, "C++")
print (myList)
# 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()**

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

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

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

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

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

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

Exemple. suppression d'une liste

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

Exemple. vider une liste

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

1.7.7 Les différente méthodes destinées aux listes Python

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

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

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

1.8 Les tuples

1.8.1 Définir un tuple en Python

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

Exemple. Création d'un tuple :

```
myTtuple = ("cartable", "cahier", "livre")
print(myTuple)
# 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 :

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

```
myTuple = ("cartable", "cahier", "livre")
for x in myTuple:
    print (x)
# 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 :

```
myTuple = ("cartable", "cahier", "livre")
if("cartable" in myTuple) :
    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 :

```
myTuple = ("cartable", "cahier", "livre")
print(len(myTuple))
# 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 :

```
myTuple = ("cartable", "cahier", "livre")
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 :

```
myTuple = ("cartable", "cahier", "livre")
del myTuple
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() :

```
myTuple = tuple (("cartable", "cahier", "livre"))
# notez les doubles parenthèses rondes
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 :

```
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 :

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

Exemple. Annuaire téléphonique

```
phoneBook = {"Majid" :"0556683531", "Tomas" :"0537773332", "Bernard" :"0668793338", "Hafid" :"
    066445566"}
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

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

Exemple. parcourt des clés d'un dictionnaire

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

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

```
stock={"Laptop":15, "Imprimante":35,"Tablette":27}

#modification de la valeur associée à la clé "Imprimante"
stock["Imprimante"]=42
print(stock)
# 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 cité ci-dessus, permet d'ajouter des éléments au dictionnaire :

Exemple. Ajouter un élément au stock

```
stock={"Laptop":15, "Imprimante":35,"Tablette":27}

# Ajout de l'élément "Ipad":18
stock["Ipad"]=18
print(stock)
# 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

```
stock={'Laptop': 15, 'Imprimante': 35, 'Tablette': 27, 'Ipad':22}

# Suppression de l'élément "Imprimante": 35
stock.pop("Imprimante")
print(stock)
# 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

```
stock={'Laptop': 15, 'Imprimante': 35, 'Tablette': 27, 'Ipad':22}

# Suppression du dernier élément
stock.popitem()
print(stock)
# 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

```
stock={'Laptop': 15, 'Imprimante': 35, 'Tablette': 27, 'Ipad':22}

# vider le dictionnaire
stock.clear()
print(stock)
# 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 quitte 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 :

```
mySet = {"Stylo", "Crayon", "Gomme"}
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

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

Exemple. vérification d'appartenance d'un élément

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

1.10.3 Longueur ou cardinal d'un ensemble Python

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

Exemple. longueur d'un ensemble python

```
mySet = {"Stylo", "Crayon", "Gomme"}
cardinal = len(mySet)
print("card(mySet) = ", cardinal)
# 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

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

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

Exemple. ajouter plusieurs éléments

```
mySet = {"Stylo", "Crayon", "Gomme"}
mySet.update (["Cahier", "Cartable", "Trousse"])
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()**

```
mySet = {"Stylo", "Crayon", "Gomme"}
mySet.remove("Crayon")
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() :**

```
mySet = {"Stylo", "Crayon", "Gomme"}
mySet.discard("Crayon")
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 **totalement 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

```
mySet = {"Stylo", "Crayon", "Gomme"}
mySet.clear()
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 oriété 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'atributs et méthodes communes.

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

2.2 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**

```
class Personne :
    def __init__(self,nom,age) :
        self.nom = nom
        self.age = age
P = Personne("Albert",27)
print("Le nom de la prsonne est : " , P.nom)
print("L'age de la personne est : " , P.age, " ans")
# affiche : Le nom de la prsonne est : Albert
#           L'age de la personne est : 27 ans
```

Exemple. classe **Rectangle**

```
class Rectangle :
    def __init__(self,L,l) :
        self.Longueur=L
       self.Largeur=l
monRectangle=Rectangle(7,5)
print("La longueur de mon rectangle est : ",monRectangle.Longueur)
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**

```
class Rectangle :
    def __init__(self,L,l) :
        self.Longueur=L
        self.Largeur=l

    # méthode qui calcule la surface
    def surface(self) :
        return self.Longueur*self.Largeur

# création d'un rectangle de longueur 7 et de largeur 5
monRectangle = Rectangle(7,5)
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

```
# -*- coding : utf-8 -*-
class myClass :
    def __init__(self) :
        pass
    # création d'une méthode statique
    @staticmethod
    def myStaticMethod() :
        print("Voici un exmple de méthode statique en Python")

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 :

```
#-*- coding : utf-8 -*-
class Personne :
    def __init__(self,nom,age) :
        self.nom = nom
        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 :**

```
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** :

```
class Student(Personne) :
```

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

```
Personne.__init__(self,nom,age)
```

Code de la **classe fille Student :**

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

Exemple. (**complet**)

```python
#-*- coding: utf-8 -*-
class Personne:
     def __init__(self,nom,age):
         self.nom = nom
         self.age=age
# La classe fille Student hérite de la classe mère Personne
class Student(Personne):
    # définition des attributs des attributs
    def __init__(self,nom,age,filiere):
        # héritage des attributs depuis la classe mère Personne
        Personne.__init__(self,nom,age)
        # ajout d'un nouvel attribut filiere à la classe fille
        self.filiere = filiere
Stud = Student("Albert",27,"math")
print("Le nom de l'étudiant  est : " ,Stud.nom)
print("L'age de l'étudiant  est : " ,Stud.age)
print("La filière de l'étudiant  est : " ,Stud.filiere)
```

Ce qui affiche après exécution :
Le nom de l'étudiant est : Albert
L'age de l'étudiant est : 27
La filière de l'étudiant est : math

2.6 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 :

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

def division(x,y):
    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 :

```
# On importe la totalité du module
from myModule import *

# On peut maintenant utiliser les fonction du module :
print("la somme de  de 7 et 8  est  : ",somme(7,8))
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 :

```
# On importe la fonction somme() du module
from myModule import somme

# On peut maintenant utiliser les fonction du module :
print("la somme de  de 7 et 8  est  : ",somme(7,8))
```

Chapitre 3

Les fichiers en Python

3.1 Le module os

Le module os permet de gérér l'arboressence 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 répértoire correspondant au chemin spécifié.

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

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

3.1.3 La méthode os.getcwd()

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

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

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 :

```
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 '+'** : ouverure 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 :

```
fichier.read()
```

Exemple. ouverture et lecture d'un fichier eistant

```
f = open("myFile.txt", 'r')
contenu = f.read() # lecture du contenu
print(contenu) # impression du contenu
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

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

Remarque 12. 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 2ème 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** :

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

Exemple. lecture séquentielle

```
f = open("myFile.txt", 'r')
s=""
for c in fs.read():
    s = s + c
print(s)
```

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

Exemple. lecture d'un fichier avec la boucle while

```
f = open("myFile.txt", 'r')
s=""
while 1:
    c = fs.read(1)
    if c =="":
       break
    s = s + c
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 nème exécution, elle pointe vers la nèmeligne.

Exemple. lecture du fichier ligne par ligne

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

```
f = open("myFile.txt", 'r')
s=""
while 1:
    ligne = f.readline()
    if(ligne == ""):
      break
    s = s + ligne
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()**

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

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

```
# -*- coding: utf-8 -*-
f = open("myFile.txt",'r')
content = f.readlines()
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**

```
# -*- coding: utf-8 -*-
f = open("myFile.txt",'r')
content = f.readlines()
n = len(content)
for i in range(0,n-1):
    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

```
# -*- coding : utf-8 -*-
f = open("myFile.txt",'r')
content = f.readlines()[2] #récupération de la deuxième ligne
result = content[9 :21] # extraction depuis le caractère qui se trouve à la position 10
    jusqu'à 20
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

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

Exemple. écrire à partir de la 6 ème position

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

Exemple. ouvrir un fichier et y ajouter du contenu :

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

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

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

Remarque 13. 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 les 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" :

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

Résultat : un nouveau fichier vide est créé !

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

```
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 Exercices

Exercice 14. Ecrire un programme Python qui permet de lire un fichier existant sur le bureau nommé **monFichier.txt.** On dois préalablement récupérer le nom d'utilisateur via la commande **os.getlogin()**

Exercice 15. 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 :**

```
>>> import os
>>> f = open("moFichier.txt",'w')
```

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

Exercice 16. En mode console, en utilisant la commande **chdir()**, changer le répertoire actuel vers le bureau (on dois 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 crée sur quel répertoire ?

Exercice 17. En utilisant les méthodes **os.rename()**, créer un programme python permettant de renommer un fichier existant sur le bureau.

Exercice 18. 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 19. Créer un programme Python permettant de remplacer les 24 premiers caractères d'un fichier existant sur le bureu 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 crée automatiquement dans le même réprtoire 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
```

```
req = "CREATE TABLE students(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL,
    email TEXT NOT NULL)"
# Exécution de la requete
cur.execute(req)
# Envoyer la requete
conn.commit()
# Fermer la connexion
conn.close
```

4.1.2 Insertion de données

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

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

Code complet :

```
# -*- coding: utf-8 -*-
import sqlite3
conn = sqlite3.connect('mabase.db')
cur = conn.cursor()
# Insérer une ligne de données
cur.execute("INSERT INTO students(`nom`,`email`) VALUES ('Albert', 'albert@gmail.com')")
# Engager l'opération
conn.commit()
# Fermer la connexion
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 !

```
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 !:

```
nom = "Albert"
email = "albert@gmail.com"
cur.execute("INSERT INTO students(`nom`,`email`) VALUES (nom, email)")
# 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 chaines à l'aide du **symbole : "?"**

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

Code complet :

```
# -*- coding: utf-8 -*-
import sqlite3
conn = sqlite3.connect('mabase.db')
nom = 'Albert'
email = 'albert@gmail.com'
age = 22
# Créer un cursor
cur = conn.cursor()
cur.execute("Insert into students (nom,email,age) values (?,?,?)",(nom, email, age))
conn.commit()
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

```
cur = conn.cursor()
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...**

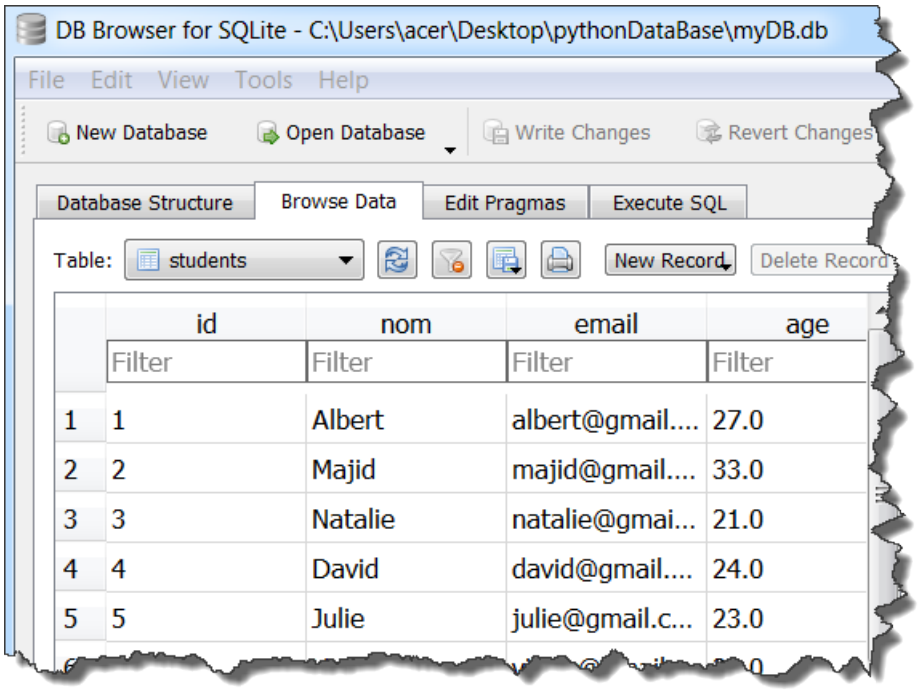
```
for row in result:
    print("ID  : ",row[0])
    print("nom  : ",row[1])
    print("Email  : ",row[2])
    print("Age  : ",row[3])
```

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 Databse**
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 :

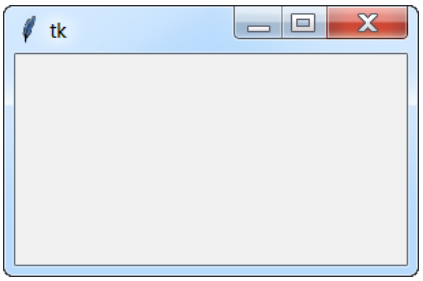
1. ***Importez le module Tkinter.***
2. ***Créez la fenêtre principale de l'application graphique.***
3. ***Ajoutez un ou plusieurs des widgets graphiques.***

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 ovales, des polygones et des rectangles, dans votre application.
3. **Checkbutton :** le widget Checkbutton permet d'afficher un certain nombre d'options sous forme de cases à cocher. L'utilisateur peut sélectionner plusieurs options à la fois.
4. **Entry :** le widget Entry est utilisé pour afficher un champ de texte d'une seule ligne permettant d'accepter les valeurs d'un utilisateur.
5. **Frame :** le widget Frame (cadre) est utilisé en tant que widget conteneur pour organiser d'autres widgets.
6. **Label :** le widget Label est utilisé pour fournir une légende ou description pour les autres widgets. Il peut aussi contenir des images.

7. **Listbox :** le widget Listbox est utilisé pour fournir une liste d'options à un utilisateur.
8. **menubutton :** le widget menubutton est utilisé pour afficher les menus dans votre application.
9. **Menu :** le widget Menu est utilisé pour fournir diverses commandes à un utilisateur. Ces commandes sont contenues dans Menubutton.
10. **Message :** le widget Message est utilisé pour afficher des champs de texte multilignes permettant d'accepter les valeurs d'un utilisateur.
11. **Radiobutton :** le widget Radiobutton est utilisé pour afficher un certain nombre d'options sous forme de boutons radio. L'utilisateur ne peut sélectionner qu'une option à la fois.
12. **Scale :** le widget Echelle est utilisé pour fournir un widget à curseur.
13. **Scrollbar :** le widget Scrollbar ou barre de défilement est utilisé pour ajouter une fonctionnalité de défilement à divers widgets, tels que les zones de liste.
14. **Text :** le widget Text est utilisé pour afficher du texte sur plusieurs lignes.
15. **Toplevel :** le widget Toplevel est utilisé pour fournir un conteneur de fenêtre séparé.
16. **Spinbox :** le widget Spinbox est une variante du widget standard **Tkinter Entry**, qui peut être utilisé pour sélectionner un nombre fixe de valeurs.
17. **PanedWindow :** le widget PanedWindow est un conteneur pouvant contenir un nombre quelconque de volets, disposés horizontalement ou verticalement.
18. **LabelFrame :** un labelframe est un simple widget de conteneur. Son objectif principal est d'agir comme un intercalaire ou un conteneur pour les dispositions de fenêtre complexes.
19. **tkMessageBox :** ce module est utilisé pour afficher des boîtes de message dans vos applications.

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 :

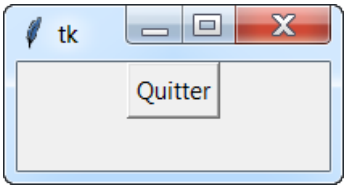
```
Nom_du_bouton = Button( Nom_de_la_fenêtre, text ="Texte du bouton", action =
    action_du_bouton())
```

Exemple. bouton quiter l'application

```
# -*- coding : utf-8 -*-
from tkinter import*
maFenetre = Tk()

#Création d'un bouton de commande
b = Button(maFenetre , text = "Quitter" , command = quit)

```

```
# Placer le bouton sur la fenêtre
b.pack()

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

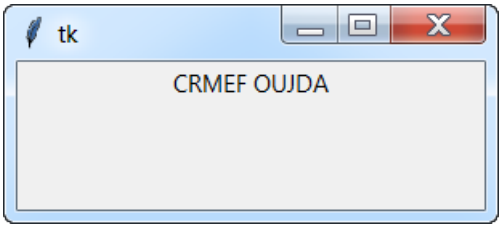
L'insertion d'un lebel sur une fenêtre Tkinter est semblable à celui d'un bouton de commande :

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

Exemple. label

```
# -*- coding : utf-8 -*-
from tkinter import*
maFenetre = Tk()

#Création du label
lbl = Label(maFenetre , text = "CRMEF OUJDA")
# Placer le label sur la fenêtre
lbl.pack()

maFenetre.mainloop()
```

Ce qui affiche après exécution :



Remarque 20. On peut aussi utiliser un texte variable, afin de donner la possibilité à l'utilisateur de modifier le texte :

Exemple. texte variable

```
# -*- coding : utf-8 -*-
from tkinter import *
fen = Tk()
var = StringVar()
```

```
label = Label( fen, textvariable=var)
var.set("CRMEF OUJDA")
label.pack()
fen.mainloop()
```

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

Exemple. ...

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

5.4.3 Le champ de saisie Entry

Le champ Entry peut servir à la récupération des données saisie par l'utilisateur afin les utiliser comme variables (formulaire d'inscription, formulaire d'identification...). La syntaxe est :

```
Nom_du_champ = Entry( Nom_de_la_fenêtre)
```

5.4.4 Le widget Text

5.4.4.1 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**

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

Le paramètre parent : représente la fenêtre parent.
options : représente la liste des options pour le widget.

5.4.4.2 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. **highlightbackbackground :** 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 évènements 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.

5.4.4.3 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 :

5.4.5 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.

5.4.5.1 Syntaxe

```
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.

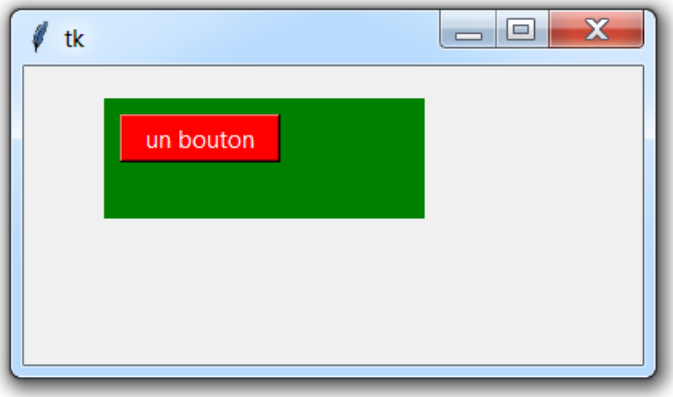
5.4.5.2 Liste des options d'un widget Frame

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

```
# -*- coding : utf-8 -*-
from tkinter import *

# Création de la fenêtre principale
master = Tk()
master.geometry("400x200")

#Création d'un frame d'arrière plan vert
frm = Frame(master, bg='green')

# Emplacement du frame
frm.place(x=50, y=20, width=200,height=75)

# Création d'un bouton au sein du frame
b = Button(frm, text="un bouton",bg='red', fg='white')
b.place(x=10, y=10, width=100,height=30) 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()**

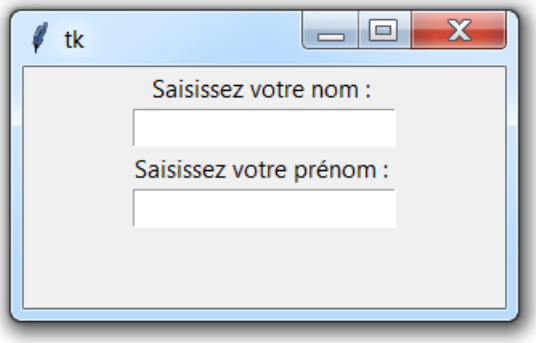
5.6.1 La méthode de disposition géométrique pack()

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 :



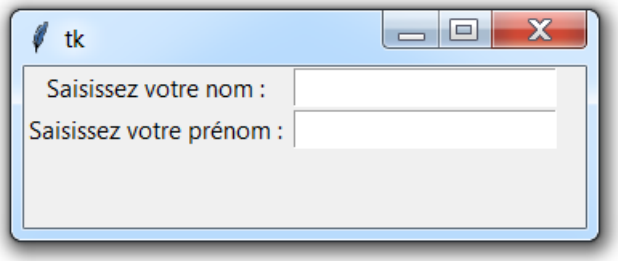
5.6.2 La méthode de disposition géométrique grid()

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()

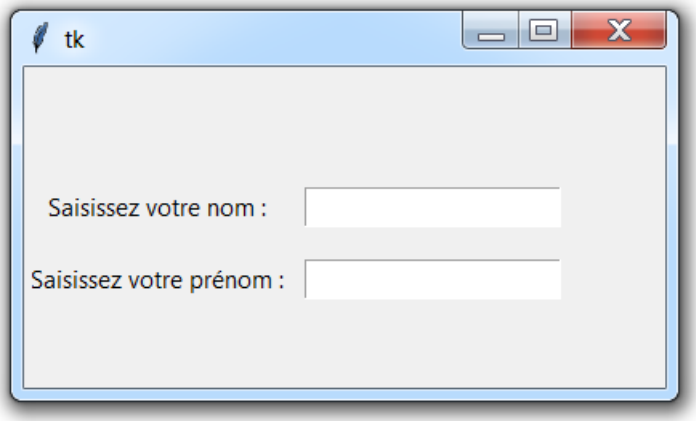
Le gestionnaire de disposition géométrique **place()**, organise les widgets en les plaçant à des positions spécifiques dans le widget parent suivant leurs coordonnées et leurs dimensions :

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
```

```
Nom = Label(f, text = "Saisissez votre nom : ")
Prenom = Label(f, text="Saisissez votre prénom : ")
champNom = Entry(f)
champPrenom = Entry(f)
#Application de la méthode place() aux widget
Nom.place(x=5, y=75, width=160, height=25)
champNom.place(x=175, y=75, width=160, height=25)
Prenom.place(x=5, y=120, width=160, height=25)
champPrenom.place(x=175, y=120, width=160, height=25)
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**

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

2. **Création d'un menu principale :**

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

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

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

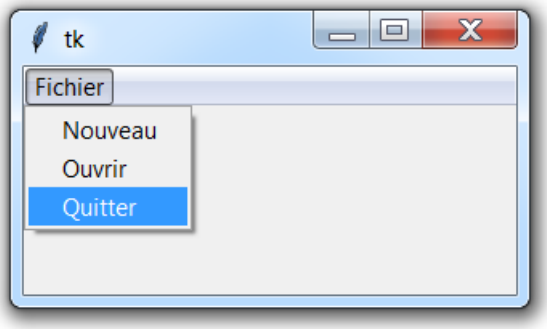
4. **Configuration de la barre des menus**

```
master.config(menu = menuBar)
```

Exemple. menu Fichier

```
# -*- coding : utf-8 -*-
from tkinter import *
master = Tk()

# Création de la barre des menu
menuBar = Menu(master)

# Création du menu principal 'Fichier'
menuFichier  = Menu(menuBar, tearoff = 0)
menuBar.add_cascade(label="Fichier",menu=menuFichier)

# Création des sous menus : 'Nouveau', 'Ouvrir', 'Quitter'
menuFichier.add_command(label="Nouveau")
menuFichier.add_command(label="Ouvrir")
menuFichier.add_command(label="Quitter", command=quit)

# Configuration de la barre des menus
master.config(menu=menuBar)
master.mainloop()
```

Ce qui affiche après exécution :



5.9 Insertion d'image sur une fenêtre Tkinter

5.10 Le module tkinter.ttk

5.10.1 A propos du module tkinter.ttk

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

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

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.

5.10.2 Usage du module tkinter.ttk

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

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

5.10.3 Usage de tkinter.ttk Button, Label & Entry

5.10.4 Usage de ttk.Combobox

5.10.5 Usage de ttk.TreeView

Références

———————————————————————

[1] Docmentation 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