Formation en langage Python

Younes. Derfoufi Enseignant au CRMEF OUJDA

23 août 2019

Table des matières

Chapitre 1

Les concepts de base 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, commentaires & opérateurs 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
```

1.3.4 Les opérateurs en Python

1.3.4.1 Les différents types d'opérateurs en Python

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

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

1.3.4.2 Les opérateurs arithmétiques

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

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

1.3.4.3 Les opérateurs d'assignation

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

Opérateurs Exemple Explication

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

1.3.4.4 Opérateurs de comparaison

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

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

1.3.4.5 Opérateurs logiques

Opérateur	Description
and	**et** logique
or	**ou** logique
not	**Négation** logique

1.4 Les fonctions en Python

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

Exemple. fonction qui renvoie le double d'un nombre

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

Remarque importante à propos de la syntaxe !

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

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

1.5 Encodage et jeux de caractères

1.6 Structures de contrôles

1.6.1 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
```

1.6.2 L'instruction elif

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

1. **Enfance age < 14**
2. **Adolescence 14 < age < 18**

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

Exemple. instruction elif

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

1.6.3 La structure répétitive For ...

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

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

Exemple. affichage des 10 premiers nombres

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

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

1.6.4 La structure répétitive While

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

1.7 Les chaines de caractères en Python

1.7.1 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
```

1.7.2 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 : 1.7.2)

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

```
s="CRMEF OUJDA"
s = s.lower()
print(s) # affiche crmef oujda
```

Exemple. remplacement d'une occurrence par une autre

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

Exemple. Nombre de caractères d'une chaine

```
#-*- coding : utf-8 -*-
s = "CRMEF OUJDA"
n = len(s)
print("le nombe de caractères de la chaine s est : " , n)
# 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
```

1.8 Les listes en Python

1.8.1 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)
```

1.8.2 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'
```

1.8.3 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']
```

1.8.4 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)
```

1.8.5 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
```

1.8.6 Ajouter ou supprimer des éléments à la liste

1.8.6.1 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"]
```

1.8.6.2 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.
```

1.8.6.3 Transformer une chaine de caractères en une liste

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

Exemple. methode split()

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

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

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

1.8.6.4 Transformer une liste en une chaine de caractères

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

Exemple. la méthode join()

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

1.8.6.5 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()**

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

1.8.7 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

1.9 Les tuples

1.9.1 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')
```

1.9.2 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.**

1.9.3 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.
```

1.9.4 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")
```

1.9.5 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
```

1.9.6 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 !
```

1.9.7 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
```

1.9.8 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)
```

1.9.9 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.

1.10 Les dictionnaires

1.10.1 Définir un dictionnaire en Python

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

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

Syntaxe :

```
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
```

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

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

Exemple. parcourt des valeurs d'un dictionnaire

```
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)
```

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

1.10.3.1 Mettre à jour un élément du dictionnaire

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

Exemple. gestionnaire d'un stock

```
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}
```

1.10.3.2 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}
```

1.10.3.3 Supprimer un élément du dictionnaire

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

Exemple. suppression d'un élément du dictionnaire

```
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}
```

1.10.3.4 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 : {}
```

1.10.4 Récapitulatif des méthodes associées à un dictionnaire

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

1. **clear() :** supprime tous les éléments du dictionnaire.
2. **copy() :** retourne une copie superficielle du dictionnaire.
3. **fromkeys(seq [, v])** : retourne un nouveau dictionnaire avec les clés de seq et une valeur égale à v (la valeur par défaut est None).
4. **get(key [, d]) :** retourne la valeur de key. Si la clé ne 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

1.11 Les ensembles Python (Python sets)

1.11.1 Définir un ensemble en Python

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

Exemple. Création d'un ensemble :

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

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

1.11.2 Accès aux éléments d'un ensemble Python

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

Exemple. Affichage des éléments d'un ensemble

```
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
```

1.11.3 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
```

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

1.11.4.1 Ajouter un ou plusieurs éléments à un ensemble Python

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

Exemple. Ajout d'un élément à l'ensemble

```
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)
```

1.11.4.2 Supprimer un élément d'un ensemble Python

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

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

```
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 !**

1.11.4.3 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
```

1.11.4.4 Supprimer un ensemble

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

Exemple. Supprimer un ensemble

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

1.11.5 Récapitulatif des méthodes associées à un ensemble Python

1. **add() :** ajoute un élément à l'ensemble
2. **clear()** : supprime tous les éléments de l'ensemble
3. **copy()** : retourne une copie de l'ensemble
4. **difference ()** : retourne un ensemble contenant la différence entre deux ensembles ou plus.
5. **difference_update() :** supprime les éléments de cet ensemble qui sont également inclus dans un autre ensemble spécifié
6. **discard() :** supprimer l'élément spécifié
7. **intersection() :** retourne un ensemble, qui est l'intersection de deux autres ensembles.
8. **intersection_update() :** supprime les éléments de cet ensemble qui ne sont pas présents dans d'autres ensembles spécifiés.
9. **isdisjoint() :** indique si deux ensembles ont une intersection ou non.
10. **issubset() :** indique si un autre jeu contient ce jeu ou non.
11. **issuperset() :** indique si cet ensemble contient un autre ensemble ou non.
12. **pop() :** supprime un élément de l'ensemble
13. **remove() :** supprime l'élément spécifié
14. **symmetric_difference() :** retourne un ensemble avec les différences symétriques de deux ensembles
15. **symmetric_difference_update() :** insère les différences symétriques de cet ensemble et d'un autre
16. **union() :** retourne un ensemble contenant l'union des ensembles
17. **update() :** met à jour l'ensemble avec l'union de cet ensemble et d'autres

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