

Les briques de base en langage Python

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

27 avril 2023

LES BRIQUES DE BASE EN LANGAGE PYTHON

Cours python
complet + 300
exercices avec
solutions

Ouvrage qui traite
le langage python
de A à Z



ÉCRIT PAR: YOUNES DERFOUFI
DOCTEUR AGRÉGÉ

Table des matières

I	Les Bases En Python	8
1	Eléments de base en Python	9
1.1	Installation des outils de développement en Python . . .	12
1.2	Premier programme en Python	16
1.3	Les variables, commentaires & opérateurs en Python . .	20
1.4	Les fonctions en Python	24
1.5	Structures de contrôles	25
1.6	Les chaînes de caractères en Python	29
1.7	Les listes en Python	36
1.8	Les tuples	41
1.9	Les tableaux (array)	44
1.10	Les dictionnaires	45
1.11	Les ensembles Python (Python sets)	49
1.12	Fonction Lambda En Python	53
1.13	Compréhension des listes en Python	55
2	Programmation orientée objet	56
2.1	Le concept de POO en Python	56
2.2	Terminologie de la POO	57
2.3	Les classes en Python	58
2.4	Les méthodes d’instances en Python	59
2.5	Les méthodes de classes en Python	60
2.6	Attributs d’instances et attributs de classes	60
2.7	Les méthodes statiques	61
2.8	Héritage en Python	61
2.9	Héritage multiple	63

2.10	Surcharge de méthodes (overloading)	63
2.11	Polymorphisme et redéfinition de méthodes (overriding methods)	65
2.12	Les classes abstraites en Python	66
2.13	Les interfaces en Python	67
3	Les modules en Python	69
3.1	Introduction	69
3.2	Créer votre propre module	69
3.3	les modules standards en Python	71
4	Exemple de modules standards	73
4.1	Le module os	73
4.2	Le module statistics	73
4.3	le module virtualenv	75
4.4	Le module PyInstaller : Transformer un script Python en un exécutable Windows	76
4.5	Le module math	77
4.6	Le module random	79
4.7	Le module collection	81
5	Les fichiers en Python	82
5.1	Le module os	82
5.2	Mode d'ouverture d'un fichier	86
5.3	Ouverture et lecture d'un fichier	87
5.4	Lecture et écriture à une position donnée à l'aide de la méthode seek()	91
5.5	Ouverture en mode écriture des fichiers en Python . . .	92
5.6	Récapitulatif des méthodes Python associées à un objet fichier avec description :	94
5.7	Manipulation des fichiers de configuration en Python . .	94
6	Python et les bases de données SQLite	101
6.1	A propos des bases de données SQLite3	101
6.2	Création de tables et de bases de données SQLite3 . . .	101
6.3	Insertion de données	103
6.4	Insertion des données de variables dans une table SQLite	103
6.5	Affichage des données d'une table SQLite3	104

6.6	Mise à jour des données SQLite3	105
6.7	Récupération totale des données sous forme d'un tableau à deux dimension	106
6.8	Exportation du contenu de la base SQLite3 vers une base sql (SQLite3 dump)	106
6.9	Éditeur WYSIWYG SQLite3	107

II Interfaces Graphiques 109

7 Les bibliothèques d'interfaces graphiques 110

7.1	Première fenêtre graphique avec Tkinter	111
7.2	Les widgets Tkinter	112
7.3	Les attributs standard des widgets Tkinter	135
7.4	Les méthodes de gestion des dispositions géométriques des widgets	136
7.5	Actions manipulant des widgets Tkinter	140
7.6	Menu Tkinter en Python	140
7.7	Les événements en Tkinter (binding event)	144
7.8	La bibliothèque d'images Pillow	149
7.9	Les boites de dialogues en Tkinter	152
7.10	Le module de design tkinter.ttk	158
7.11	Obtenir des informations sur la ligne sélectionné	169

8 Minis Projets En PythonTkinter 171

8.1	Mini Projet : Calculatrice En Python Tkinter	171
8.2	Mini projet : création d'un éditeur de texte	176
8.3	Mini Projet : logiciel de traduction	183

III Exercices Avec Solutions 190

1.	Exercices sur les bases en python : strings variables	191
2.	Exercices sur les listes Python	255
3.	Exercices sur les algorithmes design Python	317
4.	Exercices d'arithmétiques en Python	322

5. Exercices sur les dictionnaires Python	349
5. Exercices sur les ensembles Python	361
6. Exercices sur les fichiers Python	371
7. Exercices sur la programmation orientée objet en Python	385
8. Exercices sur la bibliothèque graphique Tkinter	393

A propos de l'auteur

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

trie vectorielle euclidienne, géométrie affine euclidiennes, probabilités & statistiques... Malgré tout cela, l'auteur n'a eu aucune peine à enseigner la didactiques des mathématiques et informatiques, pédagogie par objectifs PPO, méthodologie des compétences, planification des apprentissages des mathématiques et informatiques, production didactique, TICE mathématique et informatiques. L'auteur a aussi participé à de nombreux colloques & séminaires internationaux :

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

Droits d’auteur

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

Première partie

Les Bases En Python

Chapitre 1

Eléments de base en Python



1.0.1 Introduction

Python est un langage de programmation de haut niveau interprété pour la programmation à usage général. Créé par **Guido van Rossum**, et publié pour la première fois en 1991. Python repose sur une philosophie de conception qui met l'accent sur la lisibilité du code, notamment en utilisant des espaces significatifs. Il fournit des constructions permettant une programmation claire à petite et grande échelle.

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

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

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

1. **Python est utilisé par des sites web pionniers** : tels que Microsoft, YouTube, Drop Box,... Python a une forte demande sur le marché.
2. **Richesse en outils** : de nombreux IDE sont dédiés au langage Python : **Pycharm, Wing, PyScripter, Spyder...**
3. **Python est orienté objet** : la puissance du langage python est fortement marquée par son aspect orienté objet, qui permet la création et la réutilisation de codes. En raison de cette possibilité de réutilisation, le travail est effectué efficacement et réduit beaucoup de temps. Au cours des dernières années, la programmation orientée objet s'est rapporté non seulement à des classes et des objets, mais à de nombreuses bibliothèques et frameworks. Python à son tour a connu dans ce contexte un

grand essor : des **dizaines de milliers de bibliothèques** sont disponibles à l'aide de l'**outil pip de gestion des packages**.

4. **Simplicité et lisibilité du code** : Python a une syntaxe simple qui le rend approprié pour apprendre la programmation en tant que premier langage. L'apprentissage est plus fluide et rapide que d'autres langages tels que **Java**, qui nécessite très tôt une connaissance de la programmation orientée objet ou du **C/C++** qui nécessite de comprendre les pointeurs. Néanmoins, il est possible d'en apprendre davantage sur la programmation orientée objet en Python lorsqu'il est temps. Par conséquent, Python peut être utilisé comme prototype et peut être implémenté dans un autre langage de programmation après avoir testé le code.
5. **Python est open source donc gratuit** : Python étant un langage de programmation open source, il est gratuit et permet une utilisation illimitée. Avec cette licence open-source, il peut être modifié, redistribué et utilisée commercialement... Avec cette licence, Python est devenu robuste, doté de capacités évolutives et portables et est devenu un langage de programmation largement utilisé.
6. **Python est multi plateforme** : Python peut être exécuté sur tous les principaux systèmes d'exploitations, tels que : **Mac OS, Microsoft Windows, Linus et Unix...** Ce langage de programmation offre une meilleure expérience de travail avec n'importe quel système d'exploitation.
7. **Python est très puissant en terme de production** : la puissance du langage Python a été démontré sur le terrain du développement :
 - Développement Web, en utilisant les frameworks Django, Flask, Pylons
 - Science des données et visualisation à l'aide de Numpy, Pandas et Matplotlib
 - Applications de bureau avec Tkinter, PyQt, Gtk, wxWidgets et bien d'autres..
 - Applications mobiles utilisant Kivy ou BeeWare
 - Éducation : Python est un excellent langage pour apprendre l'algorithmique et la programmation ! Par conséquent largement

utilisé aux Lycées, Classes préparatoires, Instituts supérieurs, Universités...

1.1 Installation des outils de développement en Python

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

i) - Télécharger et installer Python :

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

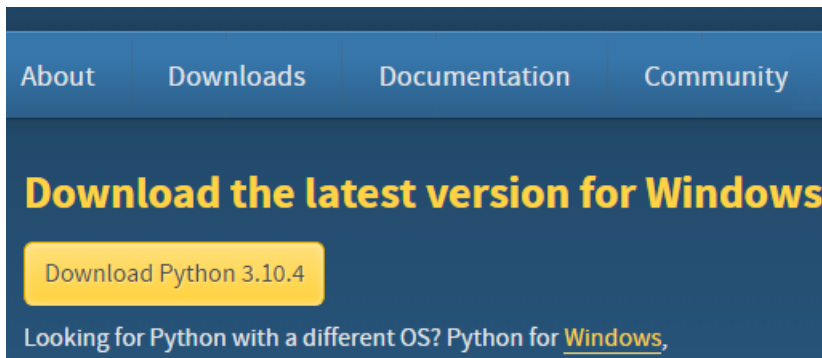
ii) - Télécharger et installer un IDE Python : de nombreux choix s'offre à vous : **Pycharm**, **PyScripter**, **Wing**. Quant à moi je vous recommande **wing**, en raison de sa rapidité et de sa simplicité d'usage, en plus il est gratuit : Télécharger Wing :

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

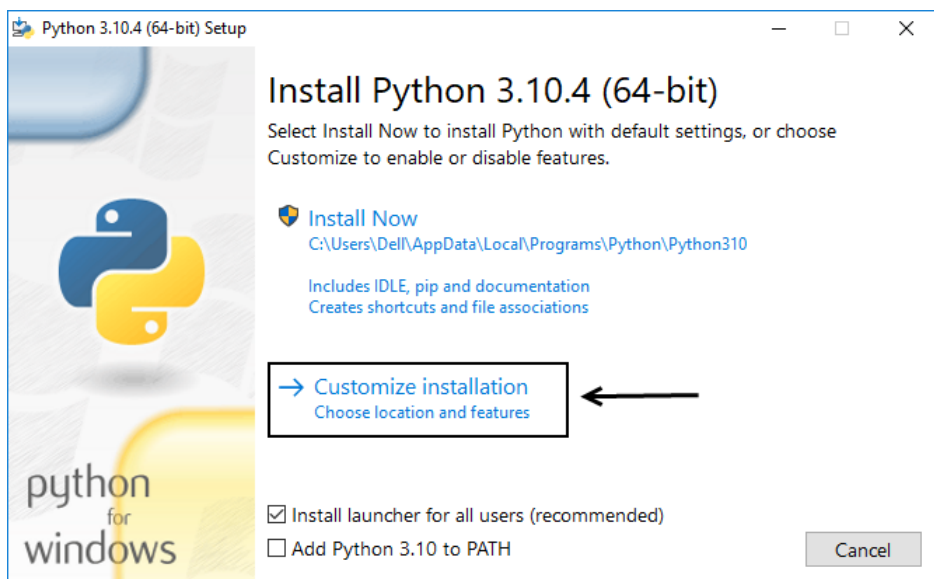
1.1.1 Installation de Python

Nous traiton ici le cas de Windows :

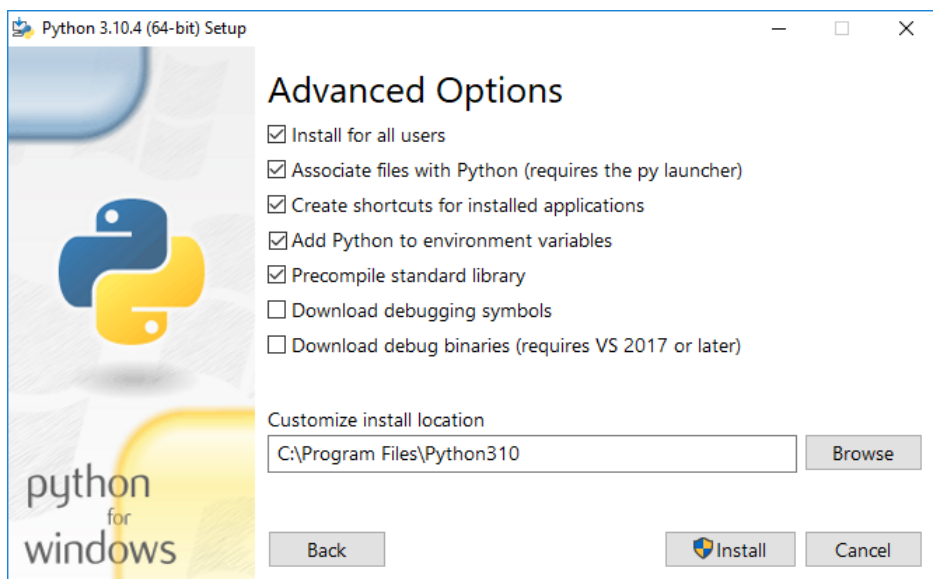
Pour télécharger l'installateur Python pour Windows, accédez à l'adresse : <https://www.python.org/downloads/> et cliquez ensuite sur le **bouton download** :



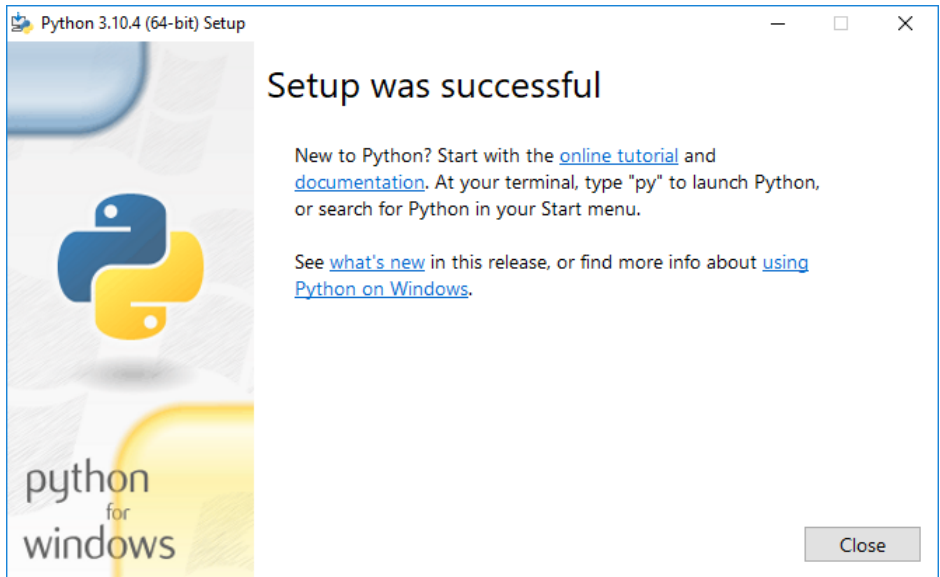
Lancer ensuite l'exécutable et cliquez sur **Customize installation** pour choisir une installation personnalisée :



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



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



1.1.2 Installation de l'IDE Wing

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

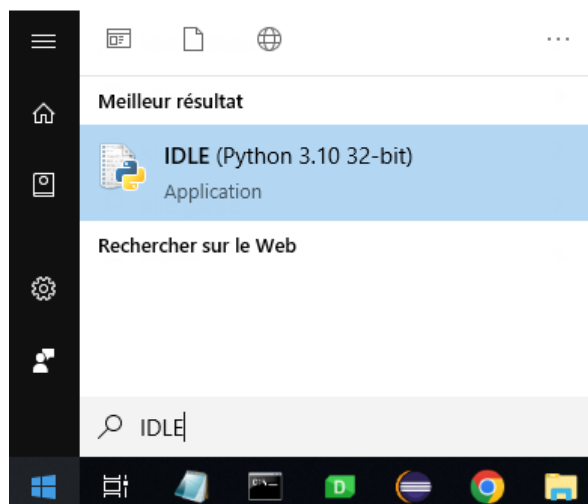


Une fois l'exécutable téléchargé, lancez le et suivez les étapes d'installation automatique par défaut.

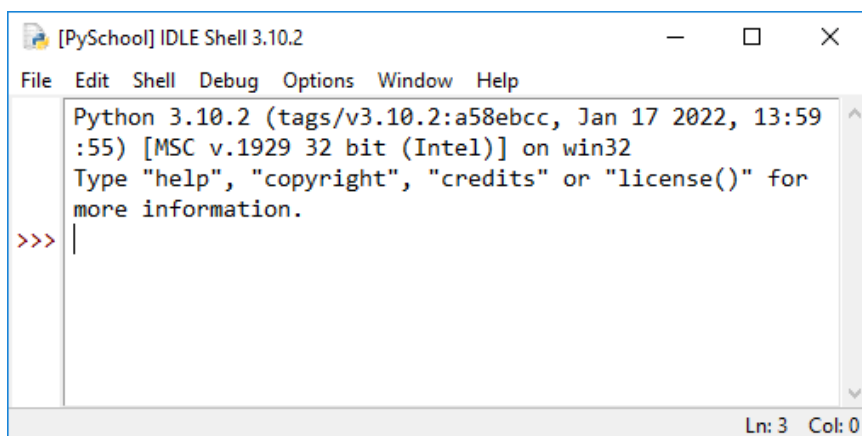
1.2 Premier programme en Python

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

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

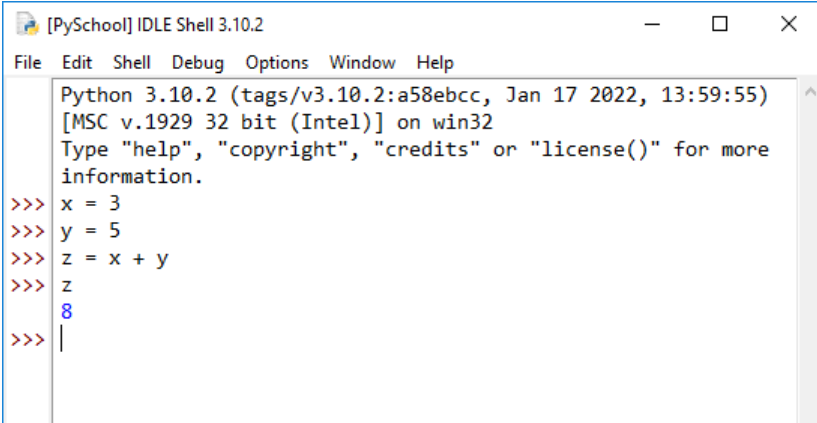


Une fois lancé, vous obtenez la vue suivante :



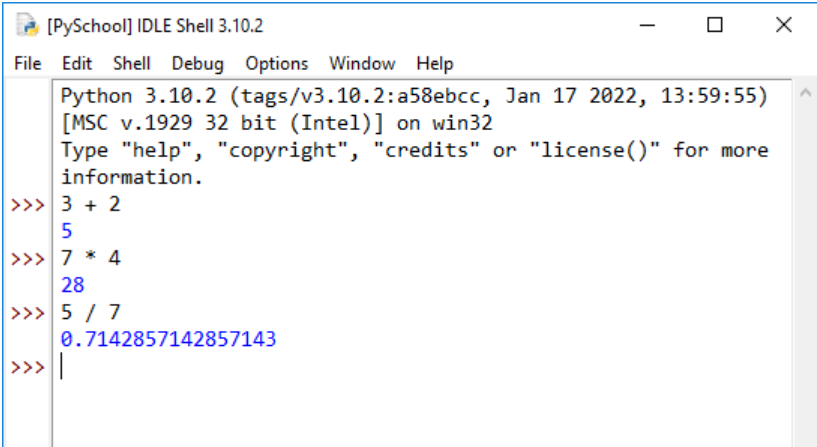
Comme vous le constatez, il s'agit d'un IDE très minimaliste. Amusons nous maintenant à faire quelques essais :

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



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

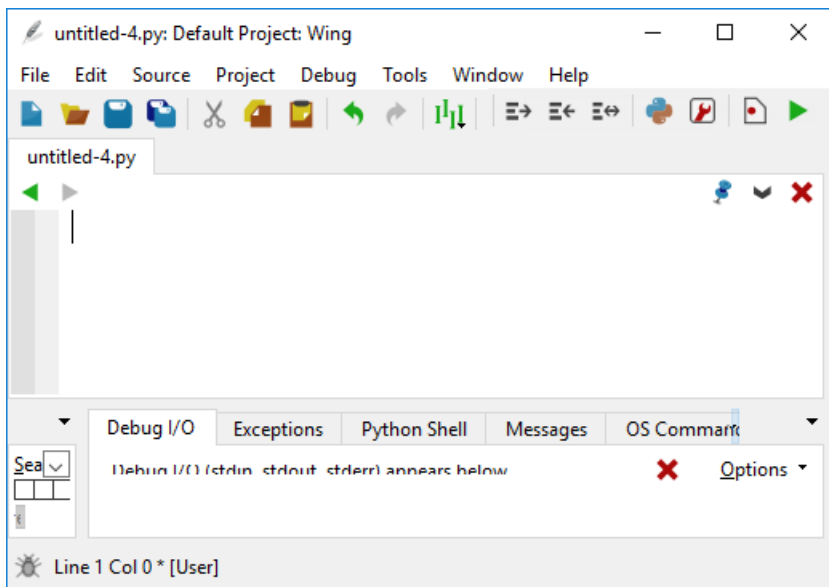
- L'outil IDLE peut aussi jouer le rôle d'une calculatrice :



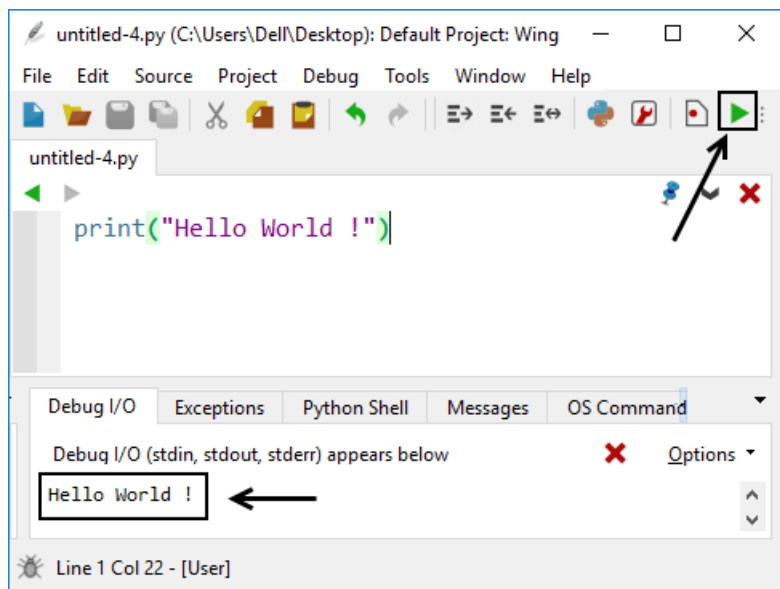
```
[PySchool] IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 13:59:55)
[MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> 3 + 2
5
>>> 7 * 4
28
>>> 5 / 7
0.7142857142857143
>>>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

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

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



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



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

1.3 Les variables, commentaires & opérateurs en Python

1.3.1 les variables en Python

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

Exemple. de variables en python

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

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

Exemple. Type d'une variable.

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

1.3.2 Affichage d'une Variable

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

Exemple. affichage variable

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

On peut aussi ajouter un texte explicatif :

Exemple. affichage avec un texte explicatif

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

1.3.3 Les commentaires en Python

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

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

1.3.3.2 Commentaire sur une seule ligne

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

Syntaxe

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

Exemple. de commentaire en python

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

1.3.3.3 Commentaire sur plusieurs lignes

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

1.3.3.4 Syntaxe

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

Exemple. commentaire sur plusieurs lignes

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


1.3.4 Les opérateurs en Python

1.3.4.1 Les différents types d'opérateurs en Python

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

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

1.3.4.2 Les opérateurs arithmétiques

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

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

1.3.4.3 Les opérateurs d'assignation

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

Opérateurs Exemple Explication

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

1.3.4.4 Opérateurs de comparaison

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

Opérateur	Description
=	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 Structures de contrôles

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

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

Syntaxe :

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

Exemple. structure if ... else...

```
1 # coding : utf-8  
2 age = 19  
3 if( age >= 18 ) :
```

```
4 print("Vous êtes majeur !")
5 else :
6     print("Vous êtes mineur !")
7 # affiche vous êtes majeur
```

1.5.2 L'instruction elif

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

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

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

Exemple. instruction **elif**

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

1.5.3 La structure répétitive For ...

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

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

Exemple. affichage des 10 premiers nombres

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

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

1.5.4 La structure répétitive While

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

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

Exemple. affichage des 10 premiers entiers avec la boucle while

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

1.5.5 Les exceptions en Python (Try Except)

1.5.5.1 Exemple introductif

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

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

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

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

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

builtins.ZeroDivisionError : division by zero.

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

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

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

1.5.5.2 Gestion des exceptions

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

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

Exemple. .

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

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

Veuillez choisir une valeur b non nulle !

1.5.5.3 Exception via une instruction raise

On peut se demander maintenant s'il est possible de lever une exception sans rencontrer une erreur. Exemple pour un programme qui demande à l'utilisateur de taper son age et de lever une exception si l'age est < 18 ans ! Bien entendu un age tapé qui est inférieur à 18 ans est une opération qui ne contient aucune erreur, et pourtant on peut quand même en lever une :

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

1.5.6 L'instruction finally

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

Syntaxe :

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

1.6 Les chaînes de caractères en Python

1.6.1 Définir une chaîne de caractère en Python

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

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

```
1 print(nom_de_la_variable_chaine)
```

Comme beaucoup d'autres langages de programmations populaires, les **chaînes de caractères** en Python sont des **tableaux d'octets** représentant des caractères Unicode. Cependant, Python ne possède pas de type de données caractère (**char**) comme char type en **C**, un

seul caractère est simplement une chaîne de longueur 1. Les crochets peuvent être utilisés pour accéder aux éléments de la chaîne.

1.6.2 Longueur d'une chaîne de caractères

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

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

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

1.6.3 Accéder aux éléments d'une chaîne de caractères

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

```
1 s[index_du_caractère]
```

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

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

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

```
1 s = "Python"
2 for i in range(0 , len(s)) :
3     print(s[i])
4 # affiche :
5 """
6 P
7 y
```



```

8 t
9 h
10 o
11 n
12 " " "

```

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

```

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

```

1.6.4 Opération sur les chaînes de caractères

1.6.4.1 Concaténation de deux chaînes de caractères

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

Exemple.

```

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

```

1.6.5 Extraire une sous chaîne de caractères

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

```

1 substring = string[i : j]

```

Exemple.

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

1.6.6 Les fonctions de chaînes de caractères en Python

Le langage Python est doté d'un grand nombre de fonctions permettant la manipulation des chaînes de caractères : calcul de la **longueur de la** chaîne, transformation en **majuscule** et **minuscule**, extraire une **sosus** chaîne...En voici une liste non exhaustive :

1. **capitalize()** : Met en majuscule la première lettre de la chaîne
2. **center(largeur, remplissage)** : Retourne une chaîne complétée par des espaces avec la chaîne d'origine centrée sur le total des colonnes de largeur.
3. **count(str, beg = 0, end = len(chaîne))** : Compte le nombre de fois où str se produit dans une chaîne ou dans une sous-chaîne si le début de l'index de début et la fin de l'index de fin sont indiqués.
4. **decode(encodage = 'UTF-8', erreurs = 'strict')** : Décode la chaîne en utilisant le codec enregistré pour le codage. Le codage par défaut correspond au codage de chaîne par défaut.
5. **encode(encoding = 'UTF-8', errors = 'strict')** : Retourne la version encodée de la chaîne; en cas d'erreur, la valeur par défaut est de générer une valeur ValueError sauf si des erreurs sont indiquées avec "ignore" ou "remplace".
6. **endswith(suffixe, début = 0, fin = len(chaîne))** : Détermine si une chaîne ou une sous-chaîne de chaîne (si les index de début et de fin d'index de fin sont indiqués) se termine par un suffixe; renvoie vrai si oui et faux sinon.
7. **expandtabs(tabsize = 8)** : Développe les onglets d'une chaîne en plusieurs espaces; La valeur par défaut est 8 espaces par onglet si tabsize n'est pas fourni.

8. **find(str, beg = 0 end = len (chaîne))** : Déterminer si str apparaît dans une chaîne ou dans une sous-chaîne de chaînes si l'index de début et l'index de fin sont spécifiés, end renvoie return s'il est trouvé et -1 dans le cas contraire.
9. **format(string s)** : remplace les accolades par la variable string s (voir exemple ci-dessous : 1.6.6)
10. **index(str, beg = 0, end = len (chaîne))** : Identique à find (), mais déclenche une exception si str n'est pas trouvé.
11. **isalnum()** : Retourne true si la chaîne a au moins 1 caractère et que tous les caractères sont alphanumériques et false sinon.
12. **isalpha()** : Retourne vrai si la chaîne a au moins 1 caractère et que tous les caractères sont alphabétiques et faux sinon.
13. **isdigit()** : Renvoie true si la chaîne ne contient que des chiffres et false sinon.
14. **islower()** : Retourne true si la chaîne a au moins 1 caractère en casse et que tous les caractères en casse sont en minuscule et false sinon.
15. **isnumeric()** : Renvoie true si une chaîne unicode contient uniquement des caractères numériques et false sinon.
16. **isspace()** : Renvoie true si la chaîne ne contient que des caractères d'espacement et false sinon.
17. **istitle()** : Retourne true si la chaîne est correctement "titlecased" et false sinon.
18. **isupper()** : Renvoie true si string contient au moins un caractère et que tous les caractères sont en majuscule et false sinon.
19. **join(seq)** : Fusionne (concatène) les représentations sous forme de chaîne d'éléments en séquence seq dans une chaîne, avec chaîne de séparation.
20. **len(chaîne)** : Retourne la longueur de la chaîne
21. **ljust(largeur [, remplissage])** : Renvoie une chaîne complétée par des espaces avec la chaîne d'origine justifiée à gauche pour un total de colonnes de largeur.
22. **lower()** : Convertit toutes les lettres majuscules d'une chaîne en minuscules.

23. **lstrip()** : Supprime tous les espaces en début de chaîne.
24. **maketrans()** : Renvoie une table de traduction à utiliser dans la fonction de traduction.
25. **max(str)** : Renvoie le caractère alphabétique maximal de la chaîne str.
26. **min(str)** : Renvoie le caractère alphabétique minimal de la chaîne str.
27. **replace(ancien, nouveau [, max])** : Remplace toutes les occurrences de old dans string par new ou au maximum max si max donné.
28. **rfind(str, beg = 0, end = len(chaîne))** : Identique à find(), mais recherche en arrière dans string.
29. **rindex(str, beg = 0, end = len (chaîne))** : Identique à index(), mais recherche en arrière dans string.
30. **rjust(largeur, [, remplissage])** : Renvoie une chaîne complétée par des espaces avec la chaîne d'origine justifiée à droite, avec un total de colonnes de largeur.
31. **rstrip()** : Supprime tous les espaces de fin de chaîne.
32. **split(str = " ", num = string.count (str))** : Divise la chaîne en fonction du délimiteur str (espace si non fourni) et renvoie la liste des sous-chaînes ; divisé en sous-chaînes au maximum, le cas échéant.
33. **splitlines(num = string.count ('\n'))** : Fractionne la chaîne de tous les NEWLINE (ou num) et renvoie une liste de chaque ligne sans les NEWLINE.
34. **startswith(str, beg = 0, end = len (chaîne))** : Détermine si string ou une sous-chaîne de chaîne (si les index de début et de fin d'index de fin sont indiqués) commence par la sous-chaîne str ; renvoie vrai si oui et faux sinon.
35. **strip([chars])** : Effectue **lstrip ()** et **rstrip ()** sur chaîne.
36. **swapcase()** : Inverse la casse de toutes les lettres d'une chaîne.
37. **title()** : Retourne la version "titlecased" de la chaîne, c'est-à-dire que tous les mots commencent par une majuscule et le reste est en minuscule.

38. **translate(table, deletechars = "")** : Traduit la chaîne en fonction de la table de traduction str (256 caractères), en supprimant celles de la chaîne del.
39. **upper()** : Convertit les lettres minuscules d'une chaîne en majuscules.
40. **zfill(largeur)** : Renvoie la chaîne d'origine laissée avec des zéros à un total de caractères de largeur ; destiné aux nombres, **zfill()** conserve tout signe donné (moins un zéro).
41. **isdecimal()** : Renvoie true si une chaîne unicode ne contient que des caractères décimaux et false sinon.
42. **s[i : j]** : Extrait la sous chaîne de s depuis la ième position jusqu'à la jème non incluse
43. **s[i :]** : Extrait la sous chaîne de s depuis la ième position jusqu'à la fin de la chaîne
44. **s[: j]** : Extrait la sous chaîne de s depuis le début jusqu'à la jème position non incluse

Exemple. transformation d'une chaîne en minuscule

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

Exemple. remplacement d'une occurrence par une autre

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

Exemple. Nombre de caractères d'une chaîne

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

Exemple. String.format

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

Exemple. extraire une sous chaîne

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

1.7 Les listes en Python

1.7.1 Création d'une liste en Python

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

Exemple. —

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

1.7.2 Accès aux éléments d'une liste.

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

Exemple. Imprimer le 3ème élément de la liste :

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

1.7.3 Changer la valeur d'un élément de la liste

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

Exemple. Changer le deuxième élément :

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

1.7.4 Parcourir les éléments d'une liste Python

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

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

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

1.7.5 Longueur d'une liste Python

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

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

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

1.7.6 Ajouter ou supprimer des éléments à la liste

1.7.6.1 Ajouter un un élément à une liste Python

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

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

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

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

Exemple. Insérer un élément en deuxième position :

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

1.7.6.2 Retirer un élément d'une liste Python

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

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

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

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

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

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


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

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

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

Exemple. suppression d'une liste

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

Exemple. vider une liste

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

1.7.6.3 Transformer une chaîne de caractères en une liste

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

Exemple. methode `split()`

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

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

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

1.7.6.4 Transformer une liste en une chaîne de caractères

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

Exemple. la méthode **join()**

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

1.7.6.5 Compter la fréquence des éléments d'une liste

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

Exemple. la méthode **Counter()**

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

1.7.7 Les différentes méthodes destinées aux listes Python

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

1. **append()** : ajoute un élément à la fin de la liste
2. **clear()** : supprime tous les éléments de la liste

3. `copy()` : retourne une copie de la liste
4. `count()` : retourne le nombre d'éléments avec la valeur spécifiée
5. `extend()` : ajoute les éléments d'une liste (ou de tout élément itérable) à la fin de la liste actuelle
6. `index()` : retourne l'index du premier élément avec la valeur spécifiée.
7. `insert()` : ajoute un élément à la position spécifiée
8. `pop()` : supprime l'élément à la position spécifiée
9. `remove()` : supprime l'élément avec la valeur spécifiée
10. `reverse()` : inverse l'ordre de la liste
11. `sort()` : trie la liste

1.8 Les tuples

1.8.1 Définir un tuple en Python

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

Exemple. Création d'un tuple :

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

1.8.2 Accéder aux élément d'un tuple

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

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

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

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