



Formation

Module I



Formation Python pour QGIS 3

Didier LECLERC
Conseiller en Management des
Systèmes d'Information Géographique
Département Relation Client

SG / SNUM / UNI / DRC



Cyril HOUISSE
Chef de projet usages et accompagnement
décisionnel, datavisualisation, datascience
Et intelligence artificielle

SG/SNUM/MSP/DS/GD3IA/PUAD



MINISTÈRE
DE LA TRANSITION
ÉCOLOGIQUE ET SOLIDAIRE
www.ecologique-solidaire.gouv.fr

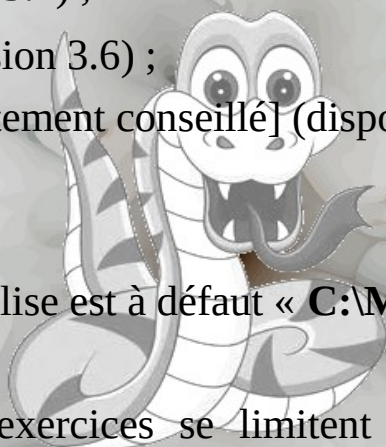
MINISTÈRE
DE LA COHÉSION
DES TERRITOIRES
www.cohesion-territoires.gouv.fr



Pré-requis logiciel :

Pour dérouler correctement cette valise, vous devez au préalable avoir installé :

- **QGIS** [obligatoire] (version 3.x) ;
- **PYTHON** [facultatif] (version 3.6) ;
- **PLUGIN RELOADER** [fortement conseillé] (disponible dans la valise – Kit)



Le répertoire d'installation de la valise est à défaut « **C:\MOC_Q3_STAGE** ».

Les jeux de données pour les exercices se limitent à ceux contenus dans le projet « **test_QGIS3.qgs** » placé dans le répertoire « **Donnees** ».

La présente valise repose sur de multiples sources documentaires disponibles sur internet, et sur l'expérience capitalisée autour de développements d'extensions pour QGIS. La plupart des sources sont indiquées. Merci à l'ensemble des contributeurs volontaires ou involontaires pour le présent support.



Conventions sur la valise :



La présence de ce pictogramme indique qu'un exercice doit être réalisé (cf. le répertoire « **Exercices** » de la valise). « **A faire** » [Obligatoire]



La présence de ce pictogramme indique qu'un exercice sera fait en démonstration par les formateurs



Zone de code
exemple

Les codes exemples

#Exemple

Les mots ou caractères importants sont en gras de couleur magenta.

Module I :

- Python, un peu d'histoire ...
- Python, généralités ...
- Python, les éditeurs ...
- Python, les caractéristiques du langage ...
- Python, premiers contacts avec le langage ...

Module II :

- Python, un langage orienté objet ...
- Python, un langage puissant ...
- Python, gérer les erreurs ...
- Python, de multiples bibliothèques et outils ...
- Python, créer des interfaces graphiques ...
- Python, aller plus loin ...



De Python à QGIS ... (*pont*)

Module III :

- QGIS, un peu d'histoire ...
- QGIS, premiers contacts avec l'API ...
- QGIS, les interfaces pour Python

Module IV :

- QGIS, créer une extension ...
- QGIS, connecter des actions ...
- QGIS, manipuler les objets des couches ...
- QGIS, aller plus loin ...



Python, avant de démarrer ...

- Pour le module **I et II**, tous nos scripts pourront être écrits et exécutés depuis **IDLE** ou depuis la console **Python de QGIS**.
- Pour le module **III, IV**, tous nos scripts pourront toujours être écrits sous **IDLE** ou dans la console **Python de QGIS**, mais ils devront être exécutés sous **QGIS**.
- *Un éditeur (NotePad++, ou PsPad), est à votre disposition pour écrire vos programmes.*
- *Ou encore, un éditeur disposant d'un débogueur par exemple*
- *Le module V, démonstration et exercice si le temps le permet, de la création d'une couche d'objets ponctuels à partir d'un référentiel linéaire sous Qgis.*

Pour un programme écrit en Python, pour l'exécuter, Il faut passer par l'interpréteur.

Exemple

```
c:/pythonxx/python.exe monprogramme.py
```

xx fait ici référence à la version de Python utilisée (ex : python26, python31, python36, ...).

Vous pouvez aussi créer un fichier « bat » pour lancer l'exécution de votre programme. Il existe aussi des « *compilateurs* » pour Python.

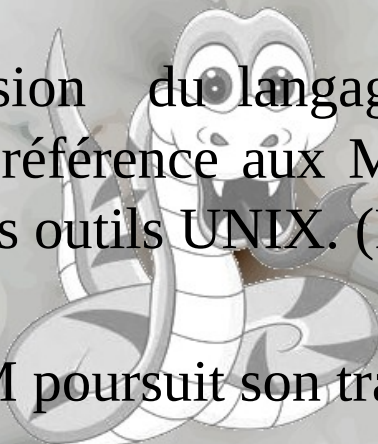


Python, un peu d'histoire ...

Concepts



- **1989** : première version du langage (**Guido VAN ROSSUM**) : langage de script, nom référence aux Monty Python, inspiré d'ABC, de Modula-3, du C et des outils UNIX. (Pays-Bas)
- **1995** : VAN ROSSUM poursuit son travail aux États-Unis.
- **1999** : Computer Programming for Everybody (CP4E) : Projet qui utilise Python comme langage d'enseignement de la programmation. Cette initiative conduira à la création de l'environnement de développement **IDLE**.



Python, un peu d'histoire ...

- **2001** : Tout code, documentation et spécification ajoutés, depuis la sortie de Python 2.1 alpha, sont détenus par la **Python Software Foundation** (PSF), une association sans but lucratif fondée en 2001, modelée d'après l'Apache Software Foundation.

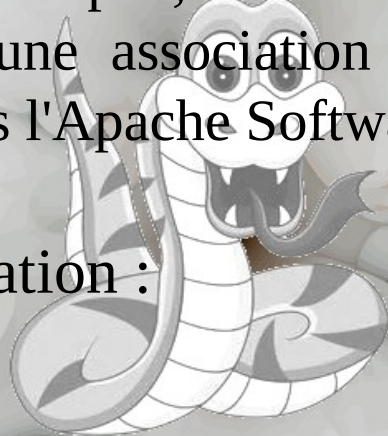
Deux sites d'information :

Site officiel :

<http://www.python.org/>

Site Association Française Python :

<http://www.afpy.org/>



Python, généralités ...

PYTHON :

Les points forts sont :

- Portable ;
- Dynamique ;
- Extensible ;
- Langage orientée objet ;
- *Gratuit (General Public License) ;*



Les points « faibles » sont :

- Langage complet et complexe, pas forcément intuitif, nombreuses API(s) complètes mais complexes, documentation en ligne exhaustive, mais difficile à s'approprier.



Python, les éditeurs ...

IDLE :

IDLE est l'IDE (« ***Integrated Development Environment*** ») Python construit avec la librairie graphique **Tkinter**. Il fait partie de la distribution de base de Python.

IDLE présente les caractéristiques suivantes:

- codé en Python pur à 100%, en utilisant la boîte à outils graphique **Tkinter** ;
- multi-plateforme: fonctionne sur Windows et Unix ;
- multi-fenêtre de l'éditeur de texte avec annulations multiples, colorisation Python et de nombreuses autres fonctionnalités, par exemple saisie intelligente (auto-complétion par la touche « ***Tabulation*** ») ;
- Python shell fenêtre (aussi connu comme interpréteur interactif) ;
- Débogueur intégré (incomplet, mais vous pouvez définir les points d'arrêt, la vue et l'étape) .

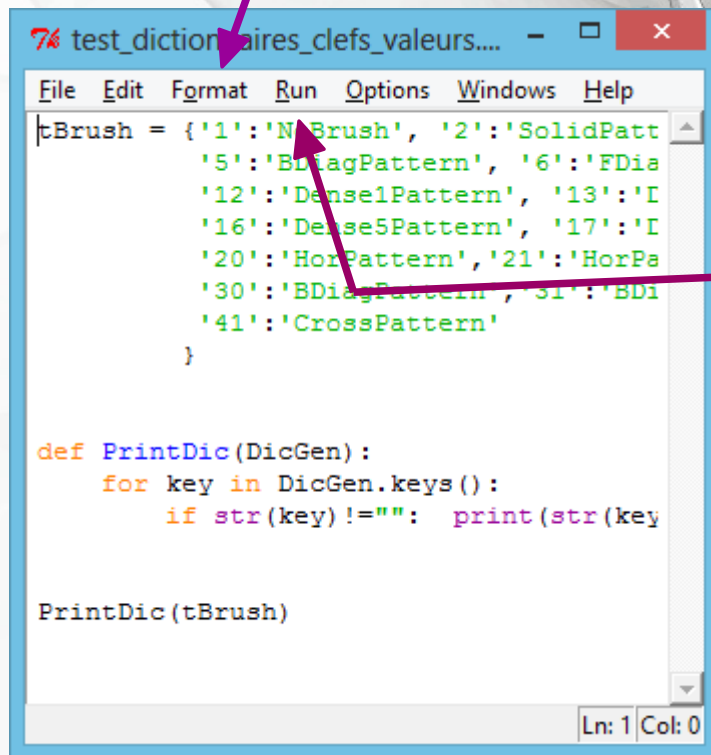
site officiel :

<http://www.python.org/idle/>



Python, les éditeurs ...

L'interface éditeur d'IDLE :



```
74 test_dictionnaires_clefs_valeurs.... - □ ×
File Edit Format Run Options Windows Help
tBrush = {'1': 'NBush', '2': 'SolidPatt
        '5': 'BDiagPattern', '6': 'FDia
        '12': 'Dense1Pattern', '13': 'I
        '16': 'Dense5Pattern', '17': 'I
        '20': 'HorPattern', '21': 'HorPa
        '30': 'BDiagPattern', '31': 'BDi
        '41': 'CrossPattern'
        }

def PrintDic(DicGen):
    for key in DicGen.keys():
        if str(key) != "": print(str(key

PrintDic(tBrush)

Ln: 1 Col: 0
```

Toutes les fonctions d'indentation, de mise en commentaires ... sont dans ce menu

Toutes les fonctions d'exécution depuis IDLE sont dans ce menu.

L'exécution de lignes de code s'obtient par la touche « **F5** » ou depuis le menu « **Run / Run Module** ».



Python, les éditeurs ...

L'interface éditeur d'IDLE :

Le résultat après exécution dans la
fenêtre d'IDLE

```
74 test_dictionnaires_clefs_valeurs.... - □ ×
File Edit Format Run Options Windows Help
tBrush = {'1':'NoBrush', '2':'SolidPatt
'5':'BDiagPattern', '6':'FDia
'12':'Dense1Pattern', '13':'I
'16':'Dense5Pattern', '17':'I
'20':'HorPattern', '21':'HorPa
'30':'BDiagPattern', '31':'BDi
'41':'CrossPattern'
}

def PrintDic(DicGen):
    for key in DicGen.keys():
        if str(key)!="": print(str(key

PrintDic(tBrush)

Ln: 1 Col: 0
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
1)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> RESTART: D:\CMSIG\3 - Formation\Python\Valise_Q3_2019\Exercices\Exo_Lister_clef
s_valeurs_dictionnaires\test_dictionnaire_did.py
ListeMonDic
Ma clef est : 1 et la valeur est : Didier
Ma clef est : 2 et la valeur est : SolidPattern
Ma clef est : 333 et la valeur est : HorPattern
Ma clef est : 4 et la valeur est : VerPattern
Ma clef est : 5 et la valeur est : BDiagPattern
Ma clef est : 6 et la valeur est : FDiagPattern
Ma clef est : 7 et la valeur est : CrossPattern
Ma clef est : 8 et la valeur est : DiagCrossPattern
Ma clef est : 12 et la valeur est : Dense1Pattern
Ma clef est : 13 et la valeur est : Dense2Pattern
Ma clef est : 14 et la valeur est : Dense3Pattern
Ma clef est : 15 et la valeur est : Dense4Pattern
Ma clef est : 16 et la valeur est : Dense5Pattern
Ma clef est : 17 et la valeur est : Dense6Pattern
Ma clef est : 18 et la valeur est : Dense7Pattern
Ma clef est : 19 et la valeur est : HorPattern
Ma clef est : 20 et la valeur est : HorPattern
Ma clef est : 21 et la valeur est : HorPattern
Ma clef est : 24 et la valeur est : VerPattern
Ma clef est : 25 et la valeur est : VerPattern
Ma clef est : 26 et la valeur est : VerPattern
Ma clef est : 29 et la valeur est : BDiagPattern
Ma clef est : 30 et la valeur est : BDiagPattern
Ma clef est : 31 et la valeur est : BDiagPattern
Ma clef est : 34 et la valeur est : FDiagPattern
Ma clef est : 35 et la valeur est : FDiagPattern
Ma clef est : 36 et la valeur est : FDiagPattern
Ma clef est : 39 et la valeur est : CrossPattern
Ma clef est : 40 et la valeur est : CrossPattern
Ma clef est : 444 et la valeur est : CrossPattern
```



Python, les éditeurs ... La console de Qgis

Concepts

La console Python de QGIS :

Zone de restitution
résultats (shell IDLE)

Exécution

Gestion par onglets de fichiers
python chargés.

The screenshot shows the QGIS Python Console window. On the left, a list of commands is displayed: 1 Console Python, 2 Utilisez iface pour accéder à l'interface de l'API QGIS ou tapez help(iface) pour plus d'informations, 3. Below this list is a prompt symbol '»'. On the right, a code editor window titled 'Exo_Lister_defs_valeurs_dictionnaires.py' contains Python code defining a dictionary 'tBrush' with 41 items. The code is color-coded and includes comments. A list of features is shown on the right side of the code editor: • Coloration syntaxique ; • Auto-complétion ; • Indentations marquées avec blocs rétractables ; • Lecture / sauvegarde « .py » ; • Commandes d'import de classes ; • Explorateur d'objets ; • Historique ; • ...

Console Python

```
1 Console Python
2 Utilisez iface pour accéder à l'inter
face de l'API QGIS ou tapez help(ifac
e) pour plus d'informations
3
```

»

Exo_Lister_defs_valeurs_dictionnaires.py

```
1 - tBrush = {'1': 'NoBrush', '2': 'SolidPattern', '3': 'HorPattern', '4': 'VerPa
2
3
4
5
6
7
8
9
10
11 - de
12 -
13
14
15
16 Pr
17
```

- Coloration syntaxique ;
- Auto-complétion ;
- Indentations marquées avec blocs rétractables ;
- Lecture / sauvegarde « .py » ;
- Commandes d'import de classes ;
- Explorateur d'objets ;
- Historique ;
- ...

Python, les caractéristiques du langage ...

Concepts

La syntaxe :

Python a été conçu pour être un langage lisible. Il vise à être visuellement épuré. Il utilise des mots anglais fréquemment là où d'autres langages utilisent de la ponctuation, et possède également moins de constructions syntaxiques que de nombreux langages structurés tels que **C**, **Perl** ou **Pascal**.

Les commentaires individuels sont indiqués par le caractère # (**hashtag**.)

```
# (c) L'auteur 2012
```

```
# propriétés {'Blanc', 'Rouge', 'Noir', 'Gris clair', 'Bleu', 'Vert foncé'}
```

Il est également possible de placer des blocs de commentaires avec la suite trois double guillemets (pour l'ouverture et la fermeture du bloc).



Python, les caractéristiques du langage ...

Il est également possible de placer des blocs de commentaires avec la suite trois double guillemets (pour l'ouverture et la fermeture du bloc).

```
'''
=====
Mon extension
=====
    Plugin pour Qgis version 3.x
=====
    Auteur       : Monsieur Deuxmaitre
    Date        : jj/mm /aaaa
    Copyright   : (C) xxxxx
    Courriel    : prenom.nom@fournisseur.fr
=====
'''

* Ce programme permet l'automatisation des traitements concernant .....
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
=====
'''
```



Python, les caractéristiques du langage ...

Concepts

Les variables :

Une variable est un espace mémoire dans lequel il est possible de mettre une valeur.

En Python, les variables sont créées automatiquement à leur première utilisation. Pour créer une variable, il suffit donc de l'utiliser en l'affectant une première fois, c'est à dire d'écrire **nom_variable = valeur_de_la_variable**.

```
x = 1      # On affecte la variable x avec la valeur du littéral 1
y = 1+1    # On affecte la variable y avec la valeur de l'expression 1 + 1
z = x + y
```

Types d'Affectations :

- Simples

x=7

- Multiples

x=y=7

Attention si on modifie la valeur de y, pas de conséquence sur x

Sauf si on affecte une liste ou un dictionnaire

- Parallèles

x,y = 7,8

- Résultante d'une fonction ou d'une opération



Python, les caractéristiques du langage ...

Concepts

Les variables :

Le nom des variables est sensible à la casse, ainsi **toto** et **Toto** ne désignent pas la même variable.

- Une variable peut prendre n'importe quel nom, tant qu'elle respecte les règles suivantes :
 - Son nom commence par une lettre minuscule (a à z) ou majuscule (A à Z), ou bien par le caractère souligné (_)
 - Pour la suite de son nom, on peut utiliser les lettres minuscules et majuscule, le souligné et un chiffre (0 à 9)
 - Son nom ne doit pas être un mot réservé.

Quelques exemples de mots réservés ou à éviter :

true, false, and, if, from, classe, lambda, def, return, ... et bien d'autres.

Par ailleurs, les noms commençant par deux « _ » et finissant par deux « _ » sont plutôt utilisés par les développeurs pour signaler la caractéristique « privée » d'une propriété ou d'une fonction.



Python, les caractéristiques du langage ...

Concepts

Les types de variables :

En Python, tout est objet.

Les types de base en **Python** sont relativement complets et puissants.
On trouve entre autres :

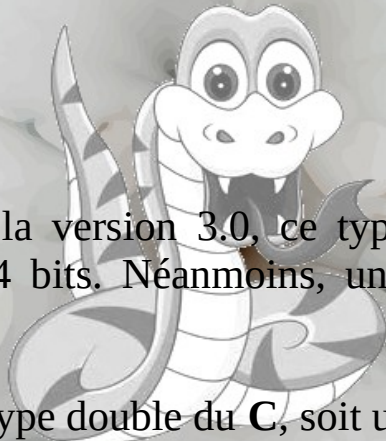
Les objets numériques :

- **int** est un entier illimité. Avant la version 3.0, ce type était dénommé long, et le type int correspondait à un entier 32 ou 64 bits. Néanmoins, une conversion automatique évitait tout débordement.
- **float** est un flottant équivalent au type double du C, soit un nombre entre $-1,7 \times 10^{308}$ et $1,7 \times 10^{308}$
- **complex** est une approximation d'un nombre complexe (typiquement deux floats)

La fonction **type()** permet de connaître le type d'une variable.

```
tFloat = 1025.758 ==> type(tFloat) Retourne « <class 'float'> »
```

```
tTexte = '1025.758' ==> type(tTexte) Retourne « <class 'str'> »
```



Python, les caractéristiques du langage ...

Concepts

Les types de variables :

Les objets « itérables » :

- Les objets **tuple** (ou n-uplet) sont des collections non mutables d'**objets** pouvant être hétérogènes.

```
MonTuple = ('Pommes', 'Poires', 100, 'Pommes')
```

- Les objets **list** sont des collections dynamiques (ils étendent automatiquement leur taille lorsque nécessaire) et acceptent des **objets** pouvant être hétérogènes.

```
MaListe = ['Pommes', 'Poires', 100, 'Pommes']
```

- `append(elt)`
- `insert(elt, index)`
- `list[index] = elt`
- `del liste[index]`
- `remove(elt)`

- Les objets **dict** sont des tableaux associatifs (ou dictionnaires) permettant d'associer une clef (un **objet**) à une « valeur » (un **objet**).

```
MonDict = {'1':'Pommes', '2':'Poires', '3':100, 'POM':'Pommes'}
```

- `dict[key] = val`
- `del dict[key]`
- `dict.pop(key)`

- Les objets **str** sont des chaînes de caractères.

```
MaChaine = 'Pommes Poires 100 Pommes'
```

- Les objets **set** sont des ensembles non ordonnés d'objets.

```
MonSet = {'Poires', 100, 'Pommes'}
```



Python, les caractéristiques du langage ...

Quelques exemples de types « itérables » :

tTuple : Je suis un objet **tuple** ! **parenthèse**

```
tTuple = ('Physique', 'Chimie', 1997, 2000)
```

str_list : Je suis un objet **list** ! **crochet**

```
str_list = [ ]
```

```
str_list.append("Pommes")
```

```
str_list.append("Poires")
```

```
str_list.append("Cerises")
```

```
str_list.append("Fraises")
```

```
str_list.append("Patates")
```

```
str_list.append("Avocats")
```

tTypeDico : Je suis un objet **dict** ! **accolade**

```
tTypeDico = {'CA':'Camemberts', 'HE':'Hémicycles', 'BA':'Histogrammes', 'HISPLUS':'Histo. empilés'}
```

tStr : Je suis un objet **str** !

```
tStr = "Exemple de chaîne de caractères, pour traitement itératif...."
```

Python, les caractéristiques du langage ...

Concepts

Les structures conditionnelles :

Python dispose des structures de contrôles classiques, à **l'exception** du « Select Case » (l'opérateur **in** apporte parfois un palliatif).

```
if cond :  
elif cond2 :  
elif cond3 :  
...  
else :
```

```
if cond :  
else :
```

Les instructions **for** et **while** sont également des structures de contrôle (type boucle).

L'instruction **break** permet d'arrêter une boucle avant sa fin.

L'instruction **continue** est similaire, mais au lieu d'interrompre la boucle, elle revient au début de celle-ci.

L'instruction **pass**, ne fait rien *En sommes-nous certain ?*



Python, les caractéristiques du langage ...

La notion de bloc et d'instruction :

Les deux points marquent le début d'instruction et les blocs sont identifiés par **l'indentation** au lieu d'accolades comme en **C**, en **C++**, en **Java** ou de **begin ... end** en **Pascal**.

Une augmentation de l'indentation marque le début d'un bloc inférieur, et une réduction de l'indentation marque la fin du bloc courant.

Les parenthèses sont facultatives dans les structures de contrôle.

Fonction factorielle en C

```
/* Fonction factorielle en C */
int factorielle(int x)
{
    if (x < 2) {
        return 1;
    }
    else {
        return x*factorielle(x-1);
    }
}
```

Fonction factorielle en Python

```
#Fonction factorielle en Python
def factorielle(x) :
    if (x < 2) :
        return 1
    else :
        return x * factorielle(x-1)
```

Les « : » marquent un début d'instruction




Python, les caractéristiques du langage ...

Concepts

Le traitement des objets « itérables » :

Les objets itérables sont parcourus à l'aide d'une boucle **for** ou **while** de la manière suivante :

Exemples :



```
for element in objet_iterable:  
    #traiter : exemple de fonction pour un traitement ...  
    traiter(element)  
  
i = 0  
while i < len(objet_iterable) :  
    #traiter : exemple de fonction pour un traitement ...  
    traiter(objet_iterable[i])  
    i += 1
```



Pour une chaîne de caractères (objet de type **str**), l'itération à défaut a lieu caractère par caractère.





Python, les caractéristiques du langage ...

Le traitement des objets « itérables » :

Rajouter un exemple des deux boucles

Exemples :

MaListe = ['Pommes', 'Poires', 100, 'Pommes']

```
for fruit in MaListe :  
    #traitement ...  
    print(fruit)
```

```
i = 0  
while i < len(MaListe) :  
    #traitement ...  
    print(MaListe[i])  
    i += 1
```

```
1 Console Python  
2 Utilisez iface pour accéder à l'interface de l'API QGIS ou  
3 >>> MaListe = ['Pommes', 'Poires', 100, 'Pommes']  
4 >>> for fruit in MaListe :  
5 ...     #traitement ...  
6 ...     print(fruit)  
7 Pommes  
8 Poires  
9 100  
10 Pommes
```

```
26 >>> while i < len(MaListe):  
27 ...     print(MaListe[i])  
28 ...     i+=1  
29 Pommes  
30 Poires  
31 100  
32 Pommes  
33
```

Démonstration



Python, les caractéristiques du langage ...

Les objets possèdent des méthodes, appelées aussi fonctions pour une classe.

Pour la classe **str** (la plus usuelle), on peut citer entre autres les fonctions suivantes pour l'objet string :

```
MaChaine = 'Pommes Poires Cerise'
```

MaChaine.**replace**(old, new, maxreplace) : remplace une sous-chaîne par une autre

MaChaine.**lstrip**() : élimine les espaces ou blancs à gauche

MaChaine.**rstrip**() : élimine les espaces ou blancs à droite

MaChaine.**strip**() : élimine tous les espaces ou blancs

MaChaine.**upper**() : met en majuscules

MaChaine.**lower**() : met en minuscules

MaChaine.**count**(carac, start, end) : compte le caractère (ou bloc de caractères)

MaChaine.**split**(carac, maxsplit) : divise en sous-blocs sur le caractère (ou bloc de caractères)

MaChaine.**join**(carac) : insère MaChaine entre chaque caractère de la chaîne carac

MaChaine.**title**() : met en titre – majuscule première lettre de chaque mot

Notez que souvent, ces fonctions sont paramétrables.

Exemple :

```
string.lstrip()      #élimine les espaces à gauche  
string.lstrip("@")  #élimine le caractère spécifié à gauche
```

Page d'information : <http://docs.python.org/2/library/string.html>



Python, les caractéristiques du langage ...

Concepts

Construire un objet de type **str** avec paramètres :

Exemple :

```
zStr = "Ma chaîne est presque vide. Pas à " + str(100) + " %"
```

Si l'on demande à visualiser le résultat de cette concaténation dans la console Python, le contenu de la variable zStr sera :

"Ma chaîne est presque vide. Pas à 100 %"

Il s'agit du formalisme classique de concaténation par l'opérateur « + ».

Pour concaténer des membres d'une liste, il sera possible d'utiliser la fonction **join** :

Exemple : notre variable de type list str_list contenant :

```
['Pommes', 'Poires', 'Cerises', 'Fraises', 'Patates', 'Avocats']
```

```
print(' '.join(str_list)) # retourne la chaîne 'Pommes Poires Cerises Fraises Patates Avocats'
```

Pour éclater une chaîne, on utilisera fonction **split** :

```
result = 'Pommes Poires Cerises Fraises Patates Avocats'
```

```
print(result.split()) # retourne la liste ['Pommes', 'Poires', 'Cerises', 'Fraises', 'Patates', 'Avocats']
```



Python, les caractéristiques du langage ...

Concepts

Manipulation de chaînes de caractères

Utilisation de la méthode de la classe **str** pour formater notre chaîne

fonction

Manipulation de la chaîne minus

Exemple :

```
minus = "bienvenue à la formation python"
```

```
minus.upper() # Je passe en majuscules avec la méthode upper()  
'BIENVENUE A LA FORMATION PYTHON'
```

```
minus.capitalize() # La première lettre en majuscule  
'Bienvenue à la formation python'
```

```
minus.title() # La première lettre de chaque mot en majuscule  
'Bienvenue Ã La Formation Python'
```

```
espaces = "    bienvenue à la formation python    "  
espaces.strip() # On retire les espaces au début et à la fin de la chaîne  
'bienvenue à la formation python'
```

```
titre = "bienvenue"  
titre.upper().center(30)  
'      BIENVENUE      '
```



Python, les caractéristiques du langage ...

Concepts

Syntaxe de la méthode **format**

Utilisation de la méthode de la classe **str** pour formater notre chaîne

nous appelons la méthode format de cette chaîne en lui passant en paramètres les variables à afficher, dans un ordre bien précis ;

Exemple :

```
#Instancier les variables
ministere1 = "MTES"
ministere2 = "MCT"
nom = "Haimar"
prenom = "jean"
age = 21
```

J'utilise la méthode «**format**»
de la variable MonMess !

```
MonMess = "Je m'appelle {3} {2}, j'ai {4} ans."
MonMess = MonMess + " Sur mon badge, il est écrit : {5} {6} – Ministère {0} / {1}"
MonMess = MonMess.format(ministere1, ministere2, nom, prenom, age,
nom.upper(),prenom.capitalize())
print(MonMess)
```

Je m'appelle jean Haimar, j'ai 21 ans. Sur mon badge, il est écrit : HAIMAR Jean – Ministère MEDDE / MLET





Python, les caractéristiques du langage ...

A vous de jouer :
Refaites cet exemple avec IDLE en 10'

Exercices

```
#Instancier les variables  
ministere1 = "MTES"  
ministere2 = "MCT"  
nom = "Haimar"  
prenom = "jean"  
age = 21
```

```
MonMess = "Je m'appelle {3} {2}, j'ai {4} ans."  
MonMess = MonMess + " Sur mon badge, il est écrit : {5} {6} – Ministère {0} / {1}"  
MonMess = MonMess.format(ministere1, ministere2, nom, prenom, age,  
nom.upper(),prenom.capitalize())  
print(MonMess)
```

Je m'appelle jean Haimar, j'ai 21 ans. Sur mon badge, il est écrit : HAIMAR Jean – Ministère MEDDE / MLET



Python, les caractéristiques du langage ...

Encore quelques exemples avec la méthode **format** et son utilisation

Exemple :

```
for i in range(3,10):  
    print(i, i*i, i*i*i)
```



```
Python 3.6.5 Shell  
File Edit Shell De  
Python 3.6.5 (v  
1)] on win32  
Type "copyright"  
>>>  
=====
```

3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729

```
>>> |
```

```
for i in range(3,10):  
    print("{:3d} {:4d} {:5d}".format(i, i*i, i*i*i))
```



```
Python 3.6.5 Shell  
File Edit Shell Debug  
Python 3.6.5 (v3.6  
1)] on win32  
Type "copyright",  
>>>  
=====
```

3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729

```
>>>
```





Python, les caractéristiques du langage ...

L'opérateur % peut-être très utile. En effet il permet de remplacer les éléments marqués par % par ceux données entre parenthèses par la suite. Le formatage des chaînes se fait via différents indicateurs, les principaux étant :

- %s : conversion en chaîne via str() avant formatage ;
- %i : nombre entier (integer) ;
- %e / %E : notation exponentielle (scientifique) d'un nombre ;
- %f / %F : nombre réel (à virgule flottante) ;
- %% : le caractère % en lui-même !

Exemple :

```
#Instancier les variables  
prenom = "jean"  
nom = "Haimar"  
age = 21  
ministere1 = "MTES"  
ministere2 = "PCT"
```

Même exemple que précédemment avec «%s»

```
MonMess = "Je m'appelle %s %s, j'ai %s ans. Sur mon badge, il est écrit : %s %s – Ministère  
%s / %s" % (prenom,nom, age, nom.upper(),prenom.capitalize(),ministere1, ministere2 )  
print(MonMess)
```

Je m'appelle jean Haimar, j'ai 21 ans. Sur mon badge, il est écrit : HAIMAR Jean – Ministère MEDDE / MLET



Python, les caractéristiques du langage ...

Définition : A la différence des données numériques, qui sont des entités singulières, les chaînes de caractères (ou string) constituent un type de donnée composite. Nous entendons par là une entité bien définie qui est faite elle-même d'un ensemble d'entités plus petites, en l'occurrence : les caractères. Suivant les circonstances, nous serons amenés à traiter une telle donnée composite, tantôt comme un seul objet, tantôt comme une suite ordonnée d'éléments. Dans ce dernier cas, nous souhaiterons probablement pouvoir accéder à chacun de ces éléments à titre individuel.

En fait, les chaînes de caractères font partie d'une catégorie d'objets Python que l'on appelle des séquences, et dont font partie aussi les listes et les tuples. On peut effectuer sur les séquences tout un ensemble d'opérations similaires. Vous en connaissez déjà quelques unes, et nous allons en décrire quelques autres.

Exemple 1 :

```
n = 'abc' + 'def'           # concaténation
m = 'zut ! ' * 4            # répétition
print n, m                  # retourne la chaîne 'abcdef zut ! zut ! zut ! zut !'
```

Exemple 2 :

```
nom = 'Cédric'
print nom[1], nom[3], nom[5] # retourne la chaîne 'é r c'
```

Sources documentaires :

http://python.developpez.com/cours/TutoSwinnen/?page=page_12



Python, les caractéristiques du langage ...

Exemple 3 :

Hachage : Dans la tranche $[n,m]$, le n ième caractère est inclus, mais pas le m ième. Si vous voulez mémoriser aisément ce mécanisme, il faut vous représenter que les indices pointent des emplacements situés entre les caractères, comme dans le schéma ci-dessous :

```
ch = "Juliette"
    ↑↑↑↑↑↑↑↑
    0 1 2 3 4 5 6 7 8
```

MaChaine = 'Juliette'

MaChaine[0:3] # retourne 'Jul' 'les 3 premiers caractères'

MaChaine[:3] # retourne 'Jul' 'les 3 premiers caractères'

MaChaine[3:] # retourne 'iette' 'tout ce qui suit les 3 premiers caractères'

MaChaine[-3:] # retourne 'tte' 'Les trois derniers caractères'

MaChaine[::-1] # retourne 'etteiluj' 'La chaîne à l'envers'

MaChaine[-5:][::-1] # retourne 'ettei' 'Les cinq derniers caractères à l'envers'



Python, les caractéristiques du langage ...

Concepts

Les fonctions :

Une fonction est un ensemble d'instructions réalisant une action.

Une fonction prend zéro, un ou plusieurs paramètres et renvoie zéro, un ou plusieurs résultats.

Une fonction est définie par le spécificateur **def** suivi du nom de la fonction et de ses paramètres.



```
def MaFonction (n):
```

```
    f = 1  
    i = 1  
    while i <= n:  
        f = f * i  
        i = i + 1
```

```
    return
```

```
def MaFonction (n):
```

```
    f = 1  
    i = 1  
    while i <= n:  
        f = f * i  
        i = i + 1
```

```
    return f
```

```
def MaFonction (n):
```

```
    f = 1  
    i = 1  
    while i <= n:  
        f = f * i  
        i = i + 1
```

```
    return f,i
```

Une fonction est utilisée de la façon suivante.

```
Mafunction(n)
```

```
F = Mafunction(n)
```

```
F, I = Mafunction(n)
```



Python, les caractéristiques du langage ...

Concepts

Exemple :

```
def ajouter(liste, valeur_a_ajouter):  
    liste.append(valeur_a_ajouter)
```

```
str_list = ['Pommes', 'Poires', 'Cerises', 'Fraises', 'Patates', 'Avocats']  
ajouter(str_list, 'Prunes')  
print(str_list)
```

Je peux accéder à la fonction
« **append** » de l'objet liste !

Source documentaire :

<http://www.siteduzero.com/tutoriel-3-281293-portee-des-variables-et-references.html>





Python, Exercices

Exercices

Créer une fonction qui va permettre de nettoyer une chaîne de caractères des « caractères » non désirés (un caractère vide se substituera au « caractère » indésirable).

Exemple :

Ma chaîne en entrée : « Voici un exemple @de ce que je souhaite réaliser@
, pour l'exemple bien@
entendu
. »

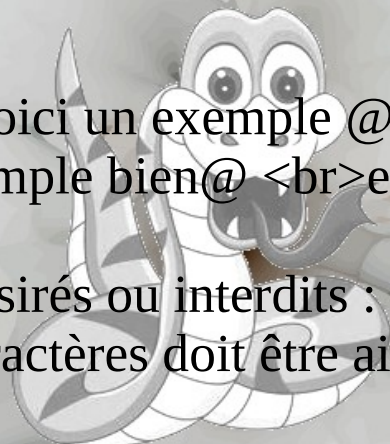
Mes « caractères » non désirés ou interdits : « @ » et «
 »

Attention : la liste des caractères doit être aisément personnalisable.

La fonction devra retourner :

« Voici un exemple de ce que je souhaite réaliser, pour l'exemple bien entendu. »

S'il vous plaît, pourrait-on améliorer et exiger que la fonction soit plus efficiente ?



Python, les caractéristiques du langage ...

Concepts

Point d'attention sur l'égalité et la référence à un objet à l'objet.

```
str_list = ['Pommes', 'Poires', 'Cerises', 'Fraises', 'Patates', 'Avocats']  
str_list2 = str_list  
str_list.append('Prunes')
```

Dans ce cas, les deux listes pointent sur le même objet ! => str_list est modifiée, str_list2 supporte la même modification.

```
#pour comparer le contenu de 2 listes  
print(str_list == str_list2) # True  
#pour comparer la référence de 2 listes  
print(str_list is str_list2) # True
```

```
str_list = ['Pommes', 'Poires', 'Cerises', 'Fraises', 'Patates', 'Avocats']  
str_list2 = list(str_list)  
#pour comparer le contenu de 2 listes  
print(str_list == str_list2) # True  
print(str_list is str_list2) # False
```

Dans ce cas, les deux listes ne pointent pas sur le même objet ! On récupère le **contenu** de la liste !

```
str_list.append('Prunes')  
#pour comparer le contenu de 2 listes  
print(str_list == str_list2) # False  
#pour comparer la référence de 2 listes  
print(str_list is str_list2) # False
```



Python, les caractéristiques du langage ...

Concepts

Un dictionnaire :

Un dictionnaire est un ensemble de couples (clés / valeurs)

On l'écrit de la façon suivante :

MonDic = {'a': 1, 'c': '3', 'b': [2,7], 'd': 4}

Quelques actions utiles:

Comment créer un dictionnaire?

```
MonDic = {}
```

Comment ajouter des valeurs dans un dictionnaire?

```
MonDic = {}
```

```
MonDic["nom"] = "engel"
```

```
MonDic["prenom"] = "olivier"
```

```
MonDic = {'nom': 'engel', 'prenom': 'olivier'}
```

Vérifier la présence d'une clé dans un dictionnaire

```
If "nom" in MonDic :
```

```
True si existe sinon False
```

Supprimer une entrée de dictionnaire

```
del MonDic["nom"]
```

Récupérer les clés ou valeurs par une boucle

```
for cle in MonDic.keys():
```

```
...     print cle
```

```
...
```

```
for valeur in MonDic.values():
```

```
...     print valeur
```

```
...
```

Récupérer les clés et les valeurs par une boucle en une seule fois

```
for cle,valeur in MonDic.items():
```

```
...     print cle, valeur
```

```
...
```

Désormais, le résultat respecte l'ordre séquentiel de construction du dictionnaire

Page d'information : <http://apprendre-python.com/page-apprendre-dictionnaire-python>



Python, les caractéristiques du langage ...

Concepts

Le traitement des objets « itérables » dictionnaire :

On peut aussi créer un dictionnaire à l'aide de l'instruction **zip** :

Exemple :

```
tuple = ('a','b','c','d')  
list = [1,2,3,4]  
dictionnaire = dict(zip(tuple,list))  
print(dictionnaire)  
#dictionnaire retourne {'a': 1, 'c': 3, 'b': 2, 'd': 4}
```





Python, Exercices

Créer une fonction « **printDic** » à partir du fichier « Exo_Lister_clefs_valeurs_dictionnaires.py » pour lister les clefs et valeurs du dictionnaire **tBrush** et obtenir le résultat suivant dans la console Python :

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:\CMSIG\3 - Formation\Python\Valise_Q3_2019\Exercices\Exo_Lister_clefs_valeurs_dictionnaires\test_dictionnaire_did.py
ListeMonDic
Ma clef est : 1 et la valeur est : Didier
Ma clef est : 2 et la valeur est : SolidPattern
Ma clef est : 333 et la valeur est : HorPattern
Ma clef est : 4 et la valeur est : VerPattern
Ma clef est : 5 et la valeur est : BDiagPattern
Ma clef est : 6 et la valeur est : FDiagPattern
Ma clef est : 7 et la valeur est : CrossPattern
Ma clef est : 8 et la valeur est : DiagCrossPattern
Ma clef est : 12 et la valeur est : Dense1Pattern
Ma clef est : 13 et la valeur est : Dense2Pattern
Ma clef est : 14 et la valeur est : Dense3Pattern
Ma clef est : 15 et la valeur est : Dense4Pattern
Ma clef est : 16 et la valeur est : Dense5Pattern
Ma clef est : 17 et la valeur est : Dense6Pattern
Ma clef est : 18 et la valeur est : Dense7Pattern
Ma clef est : 19 et la valeur est : HorPattern
Ma clef est : 20 et la valeur est : HorPattern
Ma clef est : 21 et la valeur est : HorPattern
Ma clef est : 24 et la valeur est : VerPattern
Ma clef est : 25 et la valeur est : VerPattern
Ma clef est : 26 et la valeur est : VerPattern
Ma clef est : 29 et la valeur est : BDiagPattern
Ma clef est : 30 et la valeur est : BDiagPattern
Ma clef est : 31 et la valeur est : BDiagPattern
Ma clef est : 34 et la valeur est : FDiagPattern
Ma clef est : 35 et la valeur est : FDiagPattern
Ma clef est : 36 et la valeur est : FDiagPattern
Ma clef est : 39 et la valeur est : CrossPattern
Ma clef est : 40 et la valeur est : CrossPattern
Ma clef est : 444 et la valeur est : CrossPattern
```



Allez vite au clavier ...



Python, Exercices

Exercices

Créer une fonction « **printDicTrie** » à partir du fichier « `Exo_Lister_clefs_valeurs_dictionnaires.py` » pour lister les clefs et valeurs du dictionnaire **tBrush** dans **l'ordre croissant alpha** et obtenir le résultat suivant dans la console Python :

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

ListeMonDicTrie
Ma clef est : 1 et la valeur est : Didier
Ma clef est : 2 et la valeur est : SolidPattern
Ma clef est : 4 et la valeur est : VerPattern
Ma clef est : 5 et la valeur est : BDiagPattern
Ma clef est : 6 et la valeur est : FDiagPattern
Ma clef est : 7 et la valeur est : CrossPattern
Ma clef est : 8 et la valeur est : DiagCrossPattern
Ma clef est : 12 et la valeur est : Dense1Pattern
Ma clef est : 13 et la valeur est : Dense2Pattern
Ma clef est : 14 et la valeur est : Dense3Pattern
Ma clef est : 15 et la valeur est : Dense4Pattern
Ma clef est : 16 et la valeur est : Dense5Pattern
Ma clef est : 17 et la valeur est : Dense6Pattern
Ma clef est : 18 et la valeur est : Dense7Pattern
Ma clef est : 19 et la valeur est : HorPattern
Ma clef est : 20 et la valeur est : HorPattern
Ma clef est : 21 et la valeur est : HorPattern
Ma clef est : 24 et la valeur est : VerPattern
Ma clef est : 25 et la valeur est : VerPattern
Ma clef est : 26 et la valeur est : VerPattern
Ma clef est : 29 et la valeur est : BDiagPattern
Ma clef est : 30 et la valeur est : BDiagPattern
Ma clef est : 31 et la valeur est : BDiagPattern
Ma clef est : 34 et la valeur est : FDiagPattern
Ma clef est : 35 et la valeur est : FDiagPattern
Ma clef est : 36 et la valeur est : FDiagPattern
Ma clef est : 39 et la valeur est : CrossPattern
Ma clef est : 40 et la valeur est : CrossPattern
Ma clef est : 333 et la valeur est : HorPattern
Ma clef est : 444 et la valeur est : CrossPattern
>>>
```



*Chouette,
Encore du code,
Vite, vite au clavier ...*



Python, les caractéristiques du langage ...

Concepts

La portée de variables :

L'espace « **local** » : cet espace s'entend au sens large (module).

Exemple :

```
MonNombre = 5125
def print_variable():
    print("La variable MonNombre = " + str(MonNombre))
print_variable()
```

La variable « **MonNombre** » n'est pas passée en paramètre de la fonction mais Python va la trouver.

Il est conseillé d'avoir une gestion rigoureuse des variables.

En effet, la variable « MonNombre » dans l'exemple ci-dessus n'est pas modifiable, juste consultable. Par ailleurs, une variable correctement déclarée dans une fonction sera modifiable dans cet espace local et automatiquement détruite par **Python** à la fin de l'exécution de la fonction.

une fonction ne peut modifier, par affectation, la valeur d'une variable extérieure à son espace local.



Python, les caractéristiques du langage ...

La portée de variables :

On peut définir des variables globales, toutefois il convient **de ne pas abuser**. Elles peuvent être « passées » entre modules, mais il faut préalablement importer le module qui contient la variable ou la variable elle-même.

Exemple :

```
global nOP  
nOP = 5
```

Dans un module « **options** », je crée ma variable globale !

Pour utiliser cette variable dans un autre module de mon programme, l'appel se fera sous la forme :

Exemple :

```
Import options  
if options.nOP == 5:
```

Référence explicite au module « **options** » (**import**). La variable n'est pas modifiable.



Python, les caractéristiques du langage ...

Concepts

Le style fonctionnelle :

Python permet de programmer dans un style fonctionnel.
C'est quoi un style **fonctionnel** ??

Tout est objet, même les fonctions !

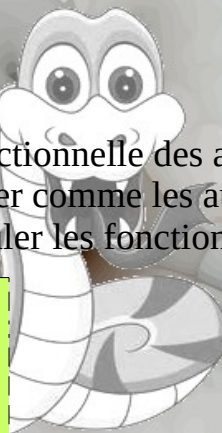
Ce qui différencie le plus la programmation fonctionnelle des autres styles, c'est probablement la notion que les fonctions sont des entités que l'on peut manipuler comme les autres dans un programme. En ce sens, le fait qu'en Python « tout est objet » nous autorise à manipuler les fonctions comme de simples variables ; on peut donc :

#Assigner une fonction à une variable

```
def FoisDeux(n):  
    return n*2
```

```
ma_fonction = FoisDeux  
ma_fonction(4)
```

8



Python, les caractéristiques du langage ...

Concepts

Le style fonctionnelle :

Python permet de programmer dans un style fonctionnel.
C'est quoi un style **fonctionnel** ??

Tout est objet, même les fonctions !

Ce qui différencie le plus la programmation fonctionnelle des autres styles, c'est probablement la notion que les fonctions sont des entités que l'on peut manipuler comme les autres dans un programme. En ce sens, le fait qu'en Python « tout est objet » nous autorise à manipuler les fonctions comme de simples variables ; on peut donc :

#Assigner une fonction à une variable

```
def FoisDeux(n):  
    return n*2
```

```
ma_fonction = FoisDeux  
ma_fonction(4)
```

8

Assigner une fonction à une variable

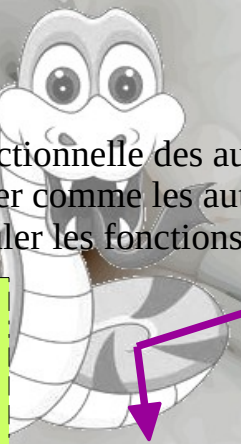
Ou encore

Passer une fonction en argument d'une autre fonction

#Passer une fonction en argument d'une autre fonction

```
def applique(fn, arg):  
    print("application de la fonction")  
    return fn(arg)
```

```
applique(FoisDeux, 4)  
application de la fonction  
8
```



Python, les caractéristiques du langage ...

Concepts

Le style fonctionnelle :

Python permet de programmer dans un style fonctionnel.
Les compréhensions de listes sont disponibles.

Par exemple, pour construire la liste des nombres pairs des 10 premiers entiers, on pourrait écrire :

```
liste = []  
  
for entier in range(10) :  
    if entier % 2 == 0 :  
        liste.append(entier)  
  
# liste = [0, 2, 4, 6, 8]  
# Pour information range(10) retourne [0,1,2,3,4,5,6,7,8,9]
```



Python, les caractéristiques du langage ...

Concepts

Le style fonctionnelle :

Python permet de programmer dans un style fonctionnel.
Les compréhensions de listes sont disponibles.

Par exemple, pour construire la liste des nombres pairs des 10 premiers entiers, on pourrait écrire :

```
liste = []  
  
for entier in range(10):  
    if entier % 2 == 0:  
        liste.append(entier)  
  
# liste = [0, 2, 4, 6, 8]
```

De façon fonctionnelle, on peut écrire l'instruction suivante :

```
liste = [entier for entier in range(10) if entier % 2 == 0]  
# liste = [0, 2, 4, 6, 8]
```



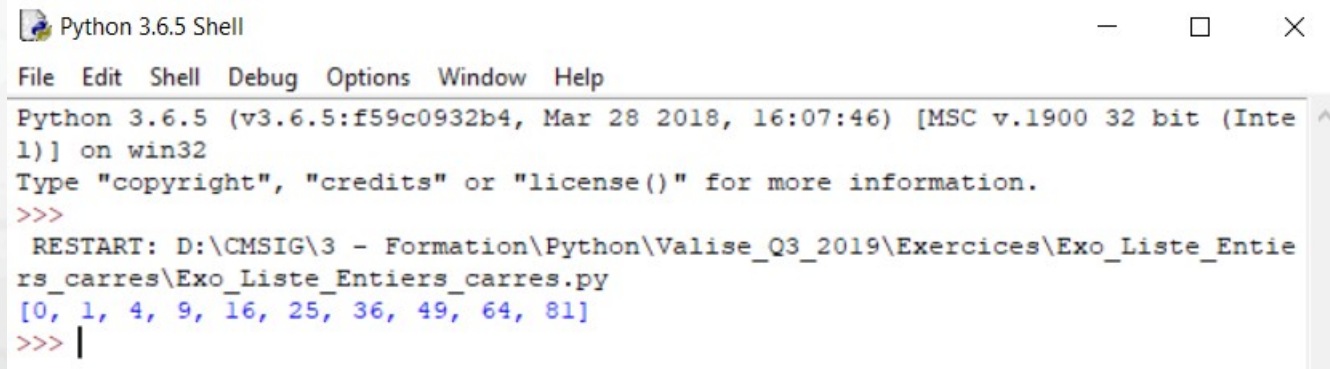


Python, Exercices

Exercices

Nous allons saisir l'instruction (en programmation fonctionnelle) permettant de créer la liste des carrés des 10 premiers entiers.

Pour visualiser le résultat (contenu de la liste), nous utiliserons l'instruction **print** (écrire le résultat dans la console Shell Python).



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:\CMSIG\3 - Formation\Python\Valise_Q3_2019\Exercices\Exo_Liste_Entiers_carres\Exo_Liste_Entiers_carres.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> |
```



Python, les caractéristiques du langage ...

Concepts

Tout est objet



La réflexivité :

Les dictionnaires sont des objets pouvant en contenir d'autres, à l'instar des listes. Un dictionnaire est un type de données extrêmement puissant et pratique. On peut placer pratiquement tout ce que l'on souhaite dans un dictionnaire, y compris des fonctions (**objet**) :

Exemple :

```
def msgError(codeError):  
    print("L'erreur " + str(codeError) + " a été trouvée !")  
def isClean(): print("Tout est Ok !")
```

```
fonctions = {}  
fonctions["msgError"] = msgError  
fonctions["isClean"] = isClean
```

Construction et alimentation du dictionnaire
(clef / « valeur »)
On ne met pas de parenthèses

```
fonctions["msgError"](69) #la présence des parenthèses → appel fonction !
```



Python, les caractéristiques du langage ...

Un module peut être appelé depuis plusieurs programmes.
N'importe quel module peut donc être appelé depuis un autre module.

Il peut contenir :

- du script
- des fonctions
- des classes
- des ressources (conversion des qrc)
- définition des variables items
- ...

```
from .bibli_autogeom import *
```

Ou

```
from . Import bibli_autogeom
```



<u>Opération:</u>	<u>Interprétation :</u>
from . <i>module</i> import <i>nom</i>	Récupère un nom (classe) particulière dans un module
from . <i>module</i> import *	Récupère tous les noms à la racine d'un module
from . import <i>module</i>	Récupère tous les noms à la racine d'un module <u>Appel d'une fonction en précisant le nom du module</u>
import <i>module</i>	Récupère un module dans son intégralité



Python, les caractéristiques du langage ...

Un exemple de syntaxe différente.

*from .math import **

Équivaut-il à ?

import math

Avec cette forme, je peux attaquer directement le nom d'une fonction !

Exemple : «pi»

Avec cette forme, je ne peux pas taper directement le nom d'une fonction !

Exemple : «math.pi »

Donc, si mon code comporte la première syntaxe et comporte des appels du type «*math.mafonction* », ça va générer des erreurs Python (phase de compilation). Idem syntaxe 2 (*import math*) avec des appels du type « *mafonction* ».

from .math import pi, sqrt

De cette façon, on peut appeler et utiliser directement la fonction pi et sqrt.



Python, les caractéristiques du langage ...

Concepts

Qu'est-ce qu'un module ? Il s'agit d'une sorte de bibliothèque (un regroupement de fonctions prédéfinies) qui une fois importée permet d'accéder à de nouvelles fonctions. Il en existe beaucoup. On peut citer :

- le module turtle qui permet de réaliser des dessins géométriques,
- le module numpy qui permet de faire du calcul scientifique,
- le module sympy qui permet de faire du calcul formel,
- le module matplotlib qui permet de faire des graphiques en tout genre.

Pour ceux que cela intéresse, il existe ce petit tutoriel pour débiter avec matplotlib .

- ceux qui sont inclus dans la version de Python comme random ou math,
- ceux que l'on peut rajouter comme numpy ou matplotlib,
- et ceux que l'on peut faire soi-même (il s'agit dans les cas simples d'un fichier Python contenant un ensemble de fonctions).

De cette façon, on peut appeler et utiliser directement la fonction pi et sqrt.

http://python.lycee.free.fr/modules_utiles.html



Python, les caractéristiques du langage ...

Concepts

Quelques rappels de bonnes pratiques

- **La casse :**

Exemple :

Str(12000) #générera une erreur

str(12000) #renvoie « 12000 »

Ce respect de la casse vaut pour tous les nommages, qu'il s'agisse de fonctions, de variables ou de résultats de tests (logiques).

- **L'indentation** (notion de blocs) ;
- **Les 'import'** des modules et appel de fonctions (utilisé dans les modules)





Formation Python pour QGIS 3



Fin du module

Merci de votre attention !!!!

