Langage Python : Les fondamentaux

I. Introduction au langage Python

Python est un langage multiplateforme, dont la première version est sortie en 1991.

Ce langage permet de développer:

- Des scripts
- Des applications lourdes
- Des sites webs (API…)

En langage Python, l'indentation est primordiale puisqu'elle détermine les blocs qui constituent votre code là où d'autres langages privilégient par exemple les accolades ({ }) pour spécifier ces blocs.

II. Commandes fondamentales

2.1. Afficher une ligne

```
print("Ceci est ma première ligne écrite en Python")
```

2.2. Demander une valeur à l'utilisateur

```
input("?")
```

2.3. Commentaires

2.3.1. Simple ligne

```
# Ce programme a été écrit par ...
```

2.3.2. Multiligne

```
'''
Ce programme a été écrit par ...
Le 15/02/2030
'''
```

III. Types de variables et opérations de base

3.1. Déclaration d'une variable

Python est un langage non-typé (ou dynamiquement typé). Autrement dit, en Python vous n'avez pas à définir explicitement le type de données que vous allez stocker dans la variable comme vous le faites en C/C++, C#, ou encore Java (qui sont des langages typés, ou fortement typés). Ici, vous pouvez directement attribuer la valeur à la variable, et le compilateur identifiera le type de données que la variable contient et la classe à laquelle elle appartient (entier, chaîne, liste, etc.).

3.2. L'opérateur +

```
print("A" + "B")
```

```
print(1 + 2)
```

Attention : la somme d'une chaine de caractère et d'un nombre n'est pas possible, il faut donc caster le nombre en chaine de caractère :

```
print("A" + str(4))
```

Ou bien passer 2 paramètres :

```
print("A", 4)
```

IV. Conditions

```
age = 35
if age < 18 :
    print("Mineur")
elif age < 25 :
    print("Etudiant")
elif age > 65 :
    print("Retraité")
else :
    print("Adulte")
```

V. Collections : Listes (tableaux 1 dimension) vs dictionnaires

5.1. Listes

5.1.1. Déclaration d'une liste

```
list = []
list = ['lundi','mardi']
list = [1, 2, 3]
list = ['mostafa','essaddouki', 31, 2019]
```

5.1.2. Manipulation d'une liste

```
list.append(1) # ajoute en fin de liste
list.insert(2, 12) # insert à la position 2
list.remove(31) # supprime l'objet 31
list.pop() # supprime le dernier élément
list.pop(1) # supprime l'élément à l'index 1
```

5.1.3. Découper la liste (slice)

```
list[:3] # 3 premiers éléments
list[3:] # à partir du 3ème élément
list[-2:] # 2 derniers éléments
list[:-2] # jusqu'à l'avant dernier élément
```

5.1.4. Fonctions utiles

```
list = [1, 6, 3, 9]
sum(list) # somme des éléments de la liste
list.count(3) # nombre d'occurences de 3
len(list) # taille de la liste
min(list) # min de tous les éléments de la liste
max(list) # max de tous les éléments de la liste
list.reverse() # Inverse l'ordre des éléments de la liste
list.sort() # Tri par ordre croissant
list.sort(reverse=True) # Tri par ordre décroissant
del list[2] # supprime l'élément à la position 2
```

5.2. Dictionnaires

5.2.1. Déclaration d'un dictionnaire

```
d = {} # dictionnaire vide

d = {
        'spam': 'eggs',
        'knights': 'lumberjack',
        'bacon': 'sausage'
}
```

VI. Boucles

6.1. Boucle While

```
x = 0
while x < 5:
    print(x)
    x += 1
```

6.2. Boucle ForEach

```
liste = [4, "banane", "tomate", 12, "raisin"]
for i in liste:
    print(i)
```

6.3. Boucle For

6.3.1. De 0 à x

```
# De 0 à 5
for i in range(6):
    print(i)
```

6.3.2. A partir d'un rang particulier

```
# De 2 à 5
for x in range(2, 6):
     print(x)
```

6.3.3. Incrémentation différente de 1

```
# De 2 à 29, par pas de 3
for x in range(2, 30, 3):
     print(x)
```

VII. Travailler avec des tableaux à 2 dimensions (DataFrame)

```python
import pandas as pd

data = {
  "fruit": ['banane', 'pomme', 'poire'],
  "calories": [89, 52, 57],
  "prix": [1.65, 2.10, 2.40]
}

df = pd.DataFrame(data)

print(df)
```

VIII. Fonctions

8.1. Procédure (fonction sans paramètre)

Une fonction python se déclare avec le mot clé **def** :

```
def dis_bonjour():
    print("Bonjour")

dis_bonjour()
```

8.2. Fonction avec paramètre

8.2.1. Paramètres classiques

```
def affiche_mon_nom(nom):
    print("Mon nom est : " + nom)

affiche_mon_nom("Nicolas");
```

8.2.2. Paramètres optionnels (*args)

```
def affiche_moi(nom, *args):
    print("Mon nom est : " + nom)
    print("Mes caractéristiques sont : ")
    for arg in args:
        print(arg)

affiche_moi ("Nicolas", "brun", "agé");
```

8.2.3. Paramètres optionnels par mot clé (*kargs)

```
def affiche_moi(nom, **kargs):
    print("Mon nom est : " + nom)
    print("Mes caractéristiques sont : ")
    for key, value in kargs.items():
        print(key + " : " + value)

affiche_moi("Nicolas", cheveux="brun", age="12");
```

```
def affiche_moi(nom, **kargs):
    print("Mon nom est : " + nom)
    print("Mes caractéristiques sont : ")
    print(kargs['cheveux'])
    print(kargs['age'])

affiche_moi("Nicolas", cheveux="brun", age="12");
```

8.3. Retour de fonction

```
def addition(n1, n2):
    return n1 + n2

r = addition(2, 3)
print(r)
```

Remarque sur la portée des variables :

- Une variable déclarée à la racine d'un module sera visible dans tout ce module. On parle de variable globale.
- Une variable déclarée dans une fonction ne sera accessible que dans cette fonction. On parle de variable locale.

IX. Classes

9.1. Déclaration

```
class Maison:

    def __init__(self, prix, surface, pays = 'France'):
        self.prix = prix
        self.surface = surface
        self.pays = pays

    def calcul_prix_metre_carre(self):
        return self.prix / self.surface
```

9.2. Instanciation

```
m = Maison(250000, 120);
print(m.calcul_prix_metre_carre())
```

X. Installer des librairies (Pip Install)

pip install *nom_librairie*

XI. Créer un environnement local (Pip Env)

11.1. Installation de pipenv

pip install pipenv

11.2. Création de l'environnement local au projet

pipenv install

11.3. Installation d'une librairie et vérification

pipenv install *nom_librairie*

pip graph

11.4. Activer / désactiver l'environnement

pipenv shell

deactivate

XII. Interroger une API (Requests)

```
import requests
response = requests.get('https://httpbin.org/ip')
print('Your IP is {0}'.format(response.json()['origin']))
```

Expressions régulières (RegEx)

12.1. Importation du module

```
import re
```

12.2. Recherche d'une chaine de caractère invariable

```
motif = "EPSI"
str = 'Je suis étudiant à l\'EPSI'

res = re.search(motif, str)

if res:
  print("Le mot correspond au motif.")
else:
  print("Le mot ne correspond pas au motif.")
```

12.3. Recherche d'une chaine de caractère flexible

```
motif = "\d{4}"
str = 'Je suis né en 1999'

res = re.search(motif, str)

if res:
  print("Date de naissance trouvée")
else:
  print("Il n'y a pas de date de naissance dans la phrase")
```

Caractère	Description	Exemple
[]	Ensemble de caractères	"[a-m]"
\	Signale une séquence spéciale (peut également être utilisé pour échapper des caractères spéciaux)	"\d"
.	N'importe quel caractère (sauf le caractère de nouvelle ligne)	"he..o"
^	Commence par	"^bonjour"
$	Finit par	"au revoir$"
*	Zero ou plus d'occurrences	"bonj.*ur"
+	Au moins une occurrence	"bonj.+ur"
?	Zero ou une occurrence (maximum)	"bonj.?ur"
{}	Exactement le nombre d'occurrences spécifié	"bon.{2}ur"
|	Ou	"bleu|rouge"

Séquence	Renvoie une correspondance pour :
\d	La chaîne contient des chiffres (nombres de 0 à 9)
\s	La chaîne contient un espace
\w	La chaîne contient des caractères de mot (de A à Z, de 0 à 9, et le caractère underscore « _ »)

Ensembles	Renvoie une correspondance pour :
[arn]	L'un des caractères spécifiés (a, r ou n) est présent
[a-n]	Tout caractère minuscule, par ordre alphabétique entre a et n
[^arn]	N'importe quel caractère SAUF a, r et n
[0123]	L'un des chiffres spécifiés (0, 1, 2 ou 3) est présent
[0-9]	Tout chiffre entre 0 et 9
[0-5][0-9]	Tous les nombres à deux chiffres entre 00 et 59
[a-zA-Z]	Tout caractère alphabétiquement entre a et z, minuscule OU majuscule
[+]	Dans les ensembles, +, *, ., |, (), $,{} n'a pas de signification particulière, donc [+] signifie : renvoie une correspondance pour tout caractère + dans la chaîne
[arn]	L'un des caractères spécifiés (a, r ou n) est présent
[a-n]	Tout caractère minuscule, par ordre alphabétique entre a et n

12.4. Groupes

Un groupe dans une expression régulière permet de récupérer certaines zones précises.

12.4.1. Résultat unique

```
regex = "(\d+)"
str = 'son numéro est le 0601020304'

res = re.search(regex, str)

if res:
    print("Numéro : " + res[0])
```

```
Numéro : 0601020304
```

12.4.2. Plusieurs résultats avec findAll()

```
regex = "([A-Z][a-z]+).+?(\w+€)"
str = 'le Kitkat est à 1€ tandis que le Kinder est à 2€'

res = re.findall(regex, str)

for art in res :
    print("L'article " + art[0] + " coûte " + art[1])
```

```
L'article Kitkat coûte 1€
L'article Kinder coûte 2€
```

12.5. Site utile pour tester ses regex

https://regex101.com/

XIII. Manipulation de fichiers

Modes	Description
r	lecture seule
r+	lecture et écriture
w	écriture uniquement
w+	écriture et lecture
a	ajouter du contenu
a+	l'ajout et la lecture
x	créer un fichier

```
fichier = open("data.txt", MODE)
fichier.close()
```

13.1. Exemples

13.1.1. Lecture

```
fichier = open("data.txt", "r")
content = fichier.read()
fichier.close()

print(content)
```

13.1.2. Ecriture

```
fichier = open("data.txt", "w")
fichier.write("J'écris dans mon fichier")
fichier.close()
```

13.2. L'instruction With

L'instruction with permet de simplifier la gestion des ressources et aide à écrire un code plus compact.

Concrètement, chaque appel à la fonction open() doit avoir un appel correspondant avec la fonction close(), cependant l'instruction with ferme automatiquement le fichier lorsque le code du bloc est exécuté.

```
with open('exemple.txt', 'r') as fichier:
    contenu = fichier.read()

print(contenu)
```

XIV. Sortie graphique

14.1. Tkinter

```
from tkinter import *

fenetre = Tk()

label = Label(fenetre, text="Bonjour")
label.pack()

fenetre.mainloop()
```

Documentation complete : https://python.doctor/page-tkinter-interface-graphique-python-tutoriel

14.2. Pygame

Documentation : https://www.zonensi.fr/Miscellanees/Pygame/Base_pygame/