Memento Python

Auteur : Antoine Pernot

• Premiers pas avec Python

Opérateurs arithmétiques :

Opération	Syntaxe
Addition	`a+b`
Soustraction	`a-b`
Multiplication	`a*b`
Division exacte	`a/b`
Division entière	`a//b`
Modulo	`a%b`
Puissance (a^b)	`a**b`
Arrondi de a, b décimales	`round(a, b)`

• Le flux d'instruction

Les conditions :

```
if condition 1:
    si la condition 1 est vraie
elif condition 2:
    si la condition 1 est fausse et la
        condition 2 est vraie
else:
    si les deux conditions sont fausses
```

Opérateurs de comparaison :

Comparateur	Syntaxe
a égal à b	`a == b`
a différent de b	`a != b`
a supérieur à b	`a > b`
a supérieur ou égal à b	`a >= b`
a inférieur à b	`a < b`
a inférieur ou égal à b	`a <= b`
a [pas] dans b	`a [not] in b`

• Factoriser le code

La boucle "Tant que"

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

Importer un module

```
import csv
from random import randint
from json import *
```

Créer une fonction

```
def addition(a, b):
    return(a+b)
```

• Les séquences

Décomposition d'une séquence

```
variable[indiceDebut:indiceFin:pas]
```

Modifier la casse Exemple : `LoReM iPsUm`

Fonction	Exemple
`lower()`	`lorem ipsum`
`upper()`	`LOREM IPSUM`
`title()`	`Lorem Ipsum`
`capitalize()`	`Lorem ipsum`
`swapcase()`	`lOrEm IpSuM`

Compter les éléments

```
sequence.count(element)
```

Remplacer dans un texte

```
chaine.replace(ancien, nouveau)
```

Ajouter un élément

```
sequence.append(element)
```

Modifier un élément

```
sequence[indice] = element
```

Insérer un élément

```
sequence[indice:indice] = [element]
```

Supprimer un élément

```
sequence.remove(element)
sequence.pop(indice ou cle)
```

Compter les éléments

```
len(sequence)
```

Éclater une chaîne en liste

```
liste = chaine.split(separateur)
```

Joindre une liste en chaîne

```
separateur.join(liste)
```

Trier une liste

```
liste.sort()
```

Inverser une liste

```
liste.reverse()
```

Mélanger une liste

```
liste.shuffle()
```

Trouver l'index d'un élément

```
liste.index(element)
```

Copier une liste

```
from copy import deepcopy
copieListe = deepcopy(liste)
```

Créer une liste de nombres

```
range(debut,fin,pas)
```

La boucle "Pour"

```
for element in liste:
    print(element)
```

Récupérer l'élément et son indice

```
for indice,element in enumerate(liste):
    print(indice,element)
```

Parcourir deux listes

```
for element1,element2 in zip(liste1,liste2
    ):
    print(element1,element2)
```

Lister les clés d'un dictionnaire

```
dictionnaire.keys()
```

Lister les valeurs d'un dictionnaire

```
dictionnaire.values()
```

Copier un dictionnaire

```
copie = dictionnaire.copy()
```

Parcourir un dictionnaire

```
for cle,valeur in dictionnaire.items():
    print(cle,valeur)
```

• Manipuler les fichiers

Navigation dans l'arborescence

```
>>> from os import getcwd, chdir, mkdir
>>> print(getcwd())
/home/antoine
>>> chdir('essais')
>>> print(getcwd())
/home/antoine/essais
>>> mkdir('test')
```

Modes de flux fichier

Lettre	Action
'r'	Ouvrir en lecture seule (défaut)
'w'	Ouvrir en écriture. Écrase l'existant.
'x'	Ouvrir en écriture si inexistant.
'a'	Ouvrir en écriture. Ajoute à l'existant.
'b'	Mode binaire.
't'	Mode texte (défaut).

Lire un fichier

```
fichier = open("texte.txt",'rt')
for ligne in fichier.read():
    print(ligne)
fichier.close()
```

Écrire un fichier

```
fichier = open("texte.txt",'wt')
fichier.write(texte)
fichier.close()
```

Lire un fichier CSV

```
import csv
fichier = open("fichier.csv", "rt")
lecteurCSV = csv.reader(fichier,delimiter=
    ";",quotechar='"')
for ligne in lecteurCSV:
    print(ligne)
fichier.close()
```

Écrire un fichier CSV

```
import csv
fichier = open("fichier.csv", "wt")
ecrivainCSV = csv.writer(fichier,
    delimiter=";")
ecrivainCSV.writerow(liste)
fichier.close()
```

Lire un fichier JSON

```
import json
fichier = open("fichier.json","rt")
dictionnaire = json.loads(fichier.read())
fichier.close()
```

Écrire un fichier JSON

```
import json
fichier = open("fichier.json","wt")
fichier.write(json.dumps(dictionnaire))
fichier.close()
```

Gestion des erreurs

```
try:
    # La portion de code à tester
except:
    # Que faire en cas d'erreur
else:
    # Si l=il n'y a pas d'erreurs
```

Les chemins de fichiers

```
>>> import os.path
>>> chemin = "/tmp/dir/dir2/fichier.txt"
>>> print(os.path.basename(chemin))
fichier.txt
>>> print(os.path.dirname(chemin))
/tmp/dir/dir2
```

Différentier les fichiers et les répertoires

Module `os.path`

Fonction	Action
exists(URI)	Si l'URI existe
isfile(URI)	Si l'URI est un fichier
isdir(URI)	Si l'URI est un dossier

Lister le contenu d'un répertoire

```
os.listdir("/tmp/dir")
```

Copier un fichier ou un répertoire

```
import shutil
shutil.copytree("dossier1","dossier2")
shutil.copy("fichier1","fichier2")
```

Déplacer un fichier ou un répertoire

```
import shutil
shutil.move("dossier1","dossier2")
```

Supprimer un fichier ou un répertoire

```
import os,shutil
os.remove("fichier")
shutil.rmtree("dossier")
```

Sauvegarder des variables

```
import pickle
fichier = open("fichier","wb")
pickle.dump(variable, fichier)
fichier.close()
```

```
import pickle
fichier = open("fichier","rb")
variable = pickle.load(fichier)
fichier.close()
```

• Interagir avec les bases de données

Exécuter une requête sur SQLite

```
import sqlite3
base = sqlite3.connect('fichier.db')
curseur = base.cursor()
curseur.execute(requete[, parametres])
base.commit()
base.close()
```

Récupérer des données sur SQLite

```
import sqlite3
base = sqlite3.connect('fichier.db')
curseur = base.cursor()
curseur.execute(requete[, parametres])
for ligne in curseur.fetchall():
    print(ligne)
base.close()
```

Exécuter une requête sur MariaDB

```
import mysql.connector
base = mysql.connector.connect(host=hote,
    user=login,password=mdp, database=base)
curseur = base.cursor()
curseur.execute(requete[, parametres])
baseDeDonnees.close()
```

Récupérer des données sur MariaDB

```
import mysql.connector
base = mysql.connector.connect(host=hote,
    user=login,password=mdp, database=base)
curseur = baseDeDonnees.cursor()
curseur.execute(requete[, parametres])
for ligne in curseur.fetchall():
    print(ligne)
```

• La programmation réseau

Créer un serveur socket

```
import socket
serveur = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
serveur.bind(('', 50000))
serveur.listen(5)
while True:
    client, infosClient = serveur.accept()
    requete = client.recv(255).decode("utf
        -8")
    client.send(reponse.encode("utf-8"))
    client.close()
serveur.close()
```

Créer un client socket

```
import socket
client = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
client.connect(("127.0.0.1", 50000))
client.send(requete.encode("utf-8"))
reponse = client.recv(255)
client.close()
```

Le multithread

```
thread = threading.Thread(None, fonction,
    nom, arguments, dictionnaireArguments)
thread.start()
```

Créer un serveur Web

```
import http.server
adresseServeur = ("", 80)
serveur = http.server.HTTPServer
handler = http.server.
    CGIHTTPRequestHandler
handler.cgi_directories = ["/tmp"]
httpd = serveur(adresseServeur, handler)
httpd.serve_forever()
```

Utiliser des services Web

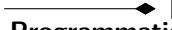
```
import urllib.request
import json
requete = urllib.request.Request(urlApi)
reponse = urllib.request.urlopen(requete)
donnees = reponse.read().decode("utf-8")
```

• Modélisation UML

Modéliser une classe

Classe
+attributPublic -attributPrive #attributProtege
+methodePublique(argument) : valeurRetour -methodePrivee() : valeurRetour (void si rien)

Relations entre classes

Lien	Relation
	Héritage
	Association
	Agrégation
	Composition

• Programmation orientée objet

Implémentation d'une classe

```
class Classe(ClasseHeritee): # object si
    aucun héritage
    def __init__(self, argument):
        # Constructeur
        ClasseHeritee.__init__(self)
        self.attributPublic
        self.__attributPrive
        variableDeLaFonction
    def __methodePrivee(self, argument):
        ...
    def methodePublique(self):
        ...
instanceClasse = Classe(argument)
instanceClasse.methodePublique()
```

Les méthodes spéciales

Méthode	Action
`__str__`	Si l'objet est appelé comme texte.
`__int__`	Si l'objet est appelé comme entier.
`__float__`	Si l'objet est appelé comme réel.

Comparaison riche

Méthode	Comparateur
`__eq__`	a égal à b
`__ne__`	a différent de b
`__gt__`	a supérieur à b
`__ge__`	a supérieur ou égal à b
`__lt__`	a inférieur à b
`__le__`	a inférieur ou égal à b