



Présentation ADRAR Pôle Numérique

Suivez-nous... LinkedIn  et Twitter  @Adrar_Numerique



ADRAR\o/ PÔLE NUMERIQUE

PÔLE NUMERIQUE DU CENTRE DE FORMATION ADRAR

- > SUPPORT, ADMINISTRATION SYSTEMES & RESEAUX
- > DEVELOPPEMENT D'APPLICATIONS WEB & MOBILES
- > WEBDESIGN & DEVELOPPEMENT INTERFACES
- > TRANSFORMATION NUMERIQUE DES ENTREPRISES

<http://www.adrar-numerique.com>

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com



Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

LES BIBLIOTHEQUES

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Une bibliothèque n'est qu'un fichier python, ou un ensemble de fichiers pythons, regroupant des classes, des fonctions, et des méthodes.

Lorsque vous importez une bibliothèque vous dites simplement à l'interpréteur que les éléments contenus dans les fichiers de cette bibliothèques peuvent être utilisés dans votre script.

Si vous pouvez importer la bibliothèque *random* c'est parcequ'il existe un fichier random.py contenant ses différentes fonctions.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Pour pouvoir utiliser une fonction ou une méthode venant d'une bibliothèque externe il faudra au préalable importer cette bibliothèque. L'import d'une bibliothèque importe également ses variables globales. On peut soit importer toute la bibliothèque, soit juste une fonction. Si on souhaite importer toute la bibliothèque on dispose de deux syntaxe, qui font la même chose mais qui vont changer la manière dont on va appeler les différents éléments de la bibliothèque.

```
import random as aleatoire
a = aleatoire.randint(0, 1)
```

import random va importer toute la bibliothèque *random* mais il faudra forcément mentionner le nom de la bibliothèque à chaque appel d'une de ses fonctions. Le mot clé *as* va permettre de donner un alias à la bibliothèque importée.

```
from random import *
a = randint(0, 1)
```

*From random import ** va importer toute la bibliothèque *random* et il n'y aura pas besoin de préfix pour appeler les fonctions. Attention cependant, vous importez également les variables sans préfixes.

```
from random import randint
a = randint(0, 1)
```

From random import randint importera uniquement la fonction *randint* du module *random*.

EXERCICES

Créez un premier fichier `donnees.py` qui contiendra une variable *TVA* qui contiendra le montant de la TVA ainsi que deux fonctions, une qui calcule le prix TTC d'un article et une autre qui calcule la somme d'un nombre indéfini de valeurs.

Créez un deuxième fichier `traitement.py` qui demandera à l'utilisateur le nombre d'article à scanner, le prix HT de ses différents articles, puis qui affichera la TVA, le prix de chaque article TTC ainsi que le prix total TTC de son panier. Ce deuxième fichier ne doit faire aucun calcul.

Au delà des fonctions que vous définissez vous-même, vous utilisez déjà régulièrement des fonctions ou des méthodes (des fonctions propres à des classes). Soit faisant directement parties de python, soit devant être importé à votre programme pour pouvoir être utilisés.

Nous allons voir l'une de ces fonctions intégrés à python et très utile : `open()`.

La fonction `open()` permet d'ouvrir un fichier texte et le stocker dans un objet de type *file* sur lequel on va pouvoir interagir. Elle prend en paramètre le nom du fichier à ouvrir ainsi que le mode d'ouverture du fichier. Il existe 4 modes:

- r : ouvre le fichier en mode lecture
- w : ouvre le fichier en mode écriture (écrase les données déjà présentes)
- a : ouvre le fichier en mode écriture (ajoute les données a la suite de celles présentes)
- x : crée le fichier

```
f = open("test.txt", "w")
```


Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Une fois que nous avons généré notre objet de type file nous allons pouvoir utiliser des méthodes pour travailler dessus, lire des lignes, en écrire,

```
f = open("test.txt", "a")
f.write("mon texte à ajouter\n")
```

La méthode write() permet d'écrire dans un fichier ouvert en mode écriture ou ajout.

```
f = open("test.txt", "r")
print(f.read())
```

La méthode read() permet d'afficher l'intégralité d'un fichier ouvert en mode lecture.

```
f = open("test.txt", "r")
print(f.readline())
```

La méthode readline() permet de lire une ligne d'un fichier ouvert en mode lecture.

```
f = open("test.txt", "r")
f.close()
```

La méthode close() permet de fermer un fichier ouvert. **IL FAUT TOUJOURS FERMER UN FICHIER OUVERT.**

Nous pouvons également utiliser une boucle for pour lire les lignes d'un fichier une par une sans utiliser de méthode en particulier.

```
f = open("test.txt", "r")  
  
for line in f:  
    print(line)  
  
f.close()
```

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique... et sur le web : www.adrar-numerique.com

Encore une fois. **IL FAUT TOUJOURS FERMER UN FICHIER OUVERT.** Un fichier ouvert peut ne pas être utilisable par un autre script ou une autre application, certaines actions de votre programme ne prendront effet qu'à la fermeture du fichier et également vous pouvez vous retrouver à atteindre la limitation de nombre de fichiers ouverts maximum (si vous parcourez un dossier avec beaucoup de fichiers et que vous les ouvrez tous par exemple)

Vous disposez du mot clé *with* qui permettra de définir un bloc d'instruction à la fin duquel le fichier sera fermé

```
with open("test.txt", "r") as f:  
    for line in f:  
        print(line)
```

EXERCICES

- Faire une fonction qui prend en paramètre un nom de fichier, qui écrit dedans 10 lignes, alternativement des lignes uniquement composées de chaînes de caractères et des lignes composées uniquement de chiffres.
- Faire une fonction qui prend en paramètre un nom de fichier et qui lit chaque ligne et qui affiche « mots » si la ligne ne contient que des chiffres, sinon elle affiche « nombres ».

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Nous allons nous intéresser au module *argparse* qui va nous permettre de définir des arguments à passer à notre script pour conditionner son fonctionnement. Exemple : un script *multi.py* qui affiche la table de multiplication d'un nombre. On peut, avec ce module, définir que l'argument `-n` ou `--nombre` définira le nombre dont on veut afficher la table de multiplication. Lorsqu'on écrira « `multi.py -n 5` » le script affichera la table de multiplication du nombre 5.

Pour savoir en détail comment fonctionne le module *argparse*, ou n'importe quel autre module, il faudra aller sur la documentation officielle : <https://docs.python.org/3/library/argparse.html> (ou comme d'habitude recherche google : python argparse)

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Le module *argparse* fonctionne de cette manière:

- On crée tout d'abord un objet de la class `ArgumentParser()`
- On appelle ensuite la méthode `add_argument()` de cet objet prenant en paramètres :
 - le nom d'appel court de l'argument
 - Le nom d'appel long de l'argument
 - Le type de valeur attendu
 - La description de l'argument à afficher à l'utilisateur quand il lancera le script avec `--help`
 - D'autres options sont possibles (référez vous à la documentation officielle)
- On crée ensuite une variable qui va contenir la totalité de nos arguments
- On rédige notre programme en indiquant les actions à effectuer si il y a tel ou tel argument

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique... et sur le web : www.adrar-numerique.com

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("-n", "--number", type=int, help="nombre dont on veut la table de multiplication")
args = parser.parse_args()

if args.number:
    for i in range(1, 10):
        print(args.number, "x", i, "=", args.number*i)
```

```
C:\Users\maximeprzybylo\Documents\Cours\Python\exercices>python multi.py --help
usage: multi.py [-h] [-n NUMBER]

options:
  -h, --help            show this help message and exit
  -n NUMBER, --number NUMBER
                        nombre dont on veut la table de multiplication
```

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique... et sur le web : www.adrar-numerique.com

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("-n", "--number", type=int, help="nombre dont on veut la table de multiplication")
args = parser.parse_args()

if args.number:
    for i in range(1, 10):
        print(args.number, "x", i, "=", args.number*i)
```

```
C:\Users\maximeprzybylo\Documents\Cours\Python\exercices>python multi.py -n 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
```

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique... et sur le web : www.adrar-numerique.com

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("-n", "--number", type=int, help="nombre dont on veut la table de multiplication")
args = parser.parse_args()

if args.number:
    for i in range(1, 10):
        print(args.number, "x", i, "=", args.number*i)
```

```
C:\Users\maximeprzybylo\Documents\Cours\Python\exercices>python multi.py --number 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
```

EXERCICES

- Faire un programme appelé touch.py qui prend en paramètre un nom et qui crée un fichier portant ce nom.
- Faire un programme appelé ls.py qui prend en paramètre un chemin de dossier et qui affiche la liste des fichiers dedans (module os)