

# Gestion des arguments en ligne de commandes et appels systèmes : exemple du find

Les notions abordées .....	2
Introduction : les arguments de ligne de commandes .....	2
Exercice 1 : argument.py .....	3
Exercice 2 : afficher le contenu d'un repertoire – find1.py .....	3
Exercice 3 : faire une recherche et stocker les repertoires et les fichiers – find2.py .....	5
Exercice 4 : un find complet – find.py .....	6
Code 1 Affichage du premier argument avec un main .....	2
Code 2 Affichage de l'ensemble des arguments via une boucle simple .....	2
Code 3 Affichage de la liste des arguments via un itérateur .....	2
Code 4 Création d'une fonction affiche avec l'argument du repertoire .....	4
Code 5 Création des listes vides .....	5
Code 6 Ajout d'éléments dans une liste .....	6
Code 7 deux arguments dans la fonction recherche .....	6
Code 8 Arguments et exemple de lancement du programme find.py .....	7
Code 9 Arborescence de test .....	7
Code 10 Exemple d'exécution du script sur une arborescence .....	7
Code 11 Structure complète de find.py .....	8
Figure 1 Exécution de argument.py .....	3
Figure 2 Exemple d'utilisation du programme find1.py .....	4
Figure 3 Affichage des variables d'environnement Windows (commande set), à noter que vous récupérer une table d'association sous Python .....	5
Figure 4 Commande find sous un système de type Unix .....	6
Figure 5 Commande dir /S sur MS Windows .....	6

## Les notions abordées

Au cours de ce TP certes très long, nous allons aborder différentes que vous avez déjà au cours des différents cours liées aux ressources de programmation :

- La structure conditionnelle if, la boucle for.
- La notion de liste et les parcours.
- Les arguments de ligne de commandes.
- Des appels systèmes qui feront appel à la notion de chemin relatif et absolu (notamment).
- Nous exécuterons ces scripts dans un IDE et dans une fenêtre d'invites de commandes (cmd ou même Powershell).

## Introduction : les arguments de ligne de commandes

Les arguments de ligne de commande sont des arguments qui viennent après la commande :

```
$> find.py -d C:/temp -f toto.txt
```

Nous avons ici un script (ou programme) avec les arguments suivants :

- `find.py` : argument 0
- `-d` : argument 1
- `C:/temp` : argument 2
- `-f` : argument 3
- `toto.txt` : argument 4

Vous retrouvez ce type d'appel souvent sous Linux mais aussi sous Powershell rendant l'appel des scripts beaucoup plus simple. Ils rendent l'appel à des programmes et des scripts beaucoup plus rapides et plus ergonomique. Ils sont doublés généralement avec des options et d'une aide. Les programmes peuvent être ensuite appelés par des scripts ou d'autres programmes.

Nous allons exploiter ces arguments durant notre TP.

Les arguments de ligne de commande se traite via un tableau d'argument. Les commandes associées sont :

- `sys.argv` : liste<sup>1</sup> d'argument qui provient du package `sys`, ainsi `sys.argv[0]` vous donne le nom du script

```
import sys
if __name__ == '__main__':
    print(sys.argv[0])
```

*Code 1 Affichage du premier argument avec un main*

- `len` : opérateur qui permet de connaître la taille du tableau, souvent utilisé pour savoir si l'utilisateur a mis le bon nombre d'arguments. Vous pouvez ainsi faire une boucle simple sur un tableau.

```
for i in range(0, len(argv)):
    print(sys.argv[i])
```

*Code 2 Affichage de l'ensemble des arguments via une boucle simple*

- Les itérateurs : oh combien pratique permettant de parcourir une liste sous une forme très simple.

```
for elt in sys.argv: # pour chaque élément de la liste sys.argv
    print(elt)
```

*Code 3 Affichage de la liste des arguments via un itérateur*

---

<sup>1</sup> La notion de tableau n'existe pas directement en Python, il s'agit de liste et non de tableau. Quelle est la différence ?

## Exercice 1 : argument.py

Coder dans un programme Python les éléments suivants :

- Un environnement d'exécution principal :

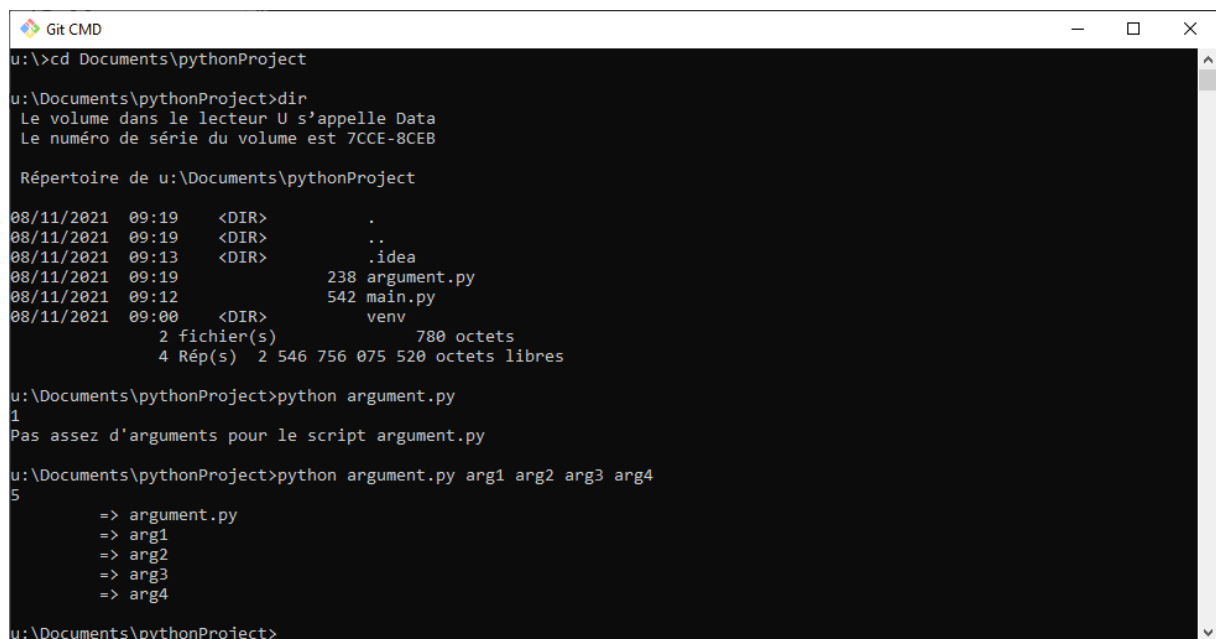
```
if __name__ == '__main__':  
    # Point d'entrée de votre programme - permet d'avoir des tests dans cette partie  
    # et pouvoir importer votre programme dans d'autres programmes  
    ...
```

Vous pouvez vous documenter sur ce point.

- Qui affiche le nombre d'arguments
- Qui vérifie le nombre d'arguments :
  - Si le nombre est égal à 1 affiche « Pas assez d'arguments pour le script <<nom du script>> <sup>2</sup>»
  - Si le nombre est supérieur strictement à 1 affiche les arguments à l'aide du code donné dans le Code 2 et le Code 3 en utilisant un `main` comme le Code 1<sup>3</sup>.

Vous pouvez ajouter des arguments de ligne de commande dans PyCharm® dans le menu Run>Edit Configurations>Votre script (argument.py)>Parameters.

Nous allons maintenant le lancer en ligne de commande<sup>4</sup>. Pour cela, vous devez savoir où vous avez enregistré votre script. Dans la capture écran ci-dessous, j'ai localisé mon script dans `U:\Documents\pythonProject` :



```
Git CMD  
u:\>cd Documents\pythonProject  
  
u:\Documents\pythonProject>dir  
Le volume dans le lecteur U s'appelle Data  
Le numéro de série du volume est 7CCE-8CEB  
  
Répertoire de u:\Documents\pythonProject  
  
08/11/2021 09:19 <DIR>      .  
08/11/2021 09:19 <DIR>      ..  
08/11/2021 09:13 <DIR>      .idea  
08/11/2021 09:19          238 argument.py  
08/11/2021 09:12          542 main.py  
08/11/2021 09:00 <DIR>      venv  
                2 fichier(s)          780 octets  
                4 Rép(s)  2 546 756 075 520 octets libres  
  
u:\Documents\pythonProject>python argument.py  
1  
Pas assez d'arguments pour le script argument.py  
  
u:\Documents\pythonProject>python argument.py arg1 arg2 arg3 arg4  
5  
=> argument.py  
=> arg1  
=> arg2  
=> arg3  
=> arg4  
  
u:\Documents\pythonProject>
```

Figure 1 Exécution de argument.py

## Exercice 2 : afficher le contenu d'un repertoire – find1.py

Il est souvent nécessaire d'accéder au contenu d'un repertoire. Vous devez connaître la notion de chemin absolu et de chemin relatif. Si vous ne connaissez pas ces notions, demandez à l'enseignant de vous l'expliquer<sup>5</sup>.

<sup>2</sup> Rappeler vous que le nom du script est donné dans l'argument 0

<sup>3</sup> Pourquoi un `main` est-il conseillé voir indispensable ? Vous pouvez regarder sur internet si nécessaire.

<sup>4</sup> Sous Linux, vous pouvez également lancer le script directement en donnant la possibilité d'exécution du script et une ligne de shebang (cela ne fonctionne pas sous Windows).

<sup>5</sup> C'est une notion importante en système que vous devez avoir compris

Pour exploiter une liste d'un répertoire, vous pouvez utiliser les commandes <sup>6</sup>suivantes :

- `os.chdir(repertoire)` : permet d'aller dans le `repertoire` correspondant
- `os.listdir(repertoire)` : permet d'obtenir une liste de fichiers et répertoires contenu dans `repertoire`.
- `os.environ[nom_variable]` : liste des variables d'environnement que vous pouvez consulter dans tapant dans un CMS : set (voir Figure 3)
  - Par exemple, dans un script python, vous exécutez `os.environ["USSENAME"]`, vous obtenez votre nom d'utilisateur (eXXXXXXX). Si vous exécutez `os.environ["USSENAME"]`, vous obtenez votre nom d'utilisateur (eXXXXXXX)
- `os.path.exists("repertoire")` : permet de vérifier l'existence du `repertoire`

Dans le script `find1.py`, effectuer les commandes suivantes :

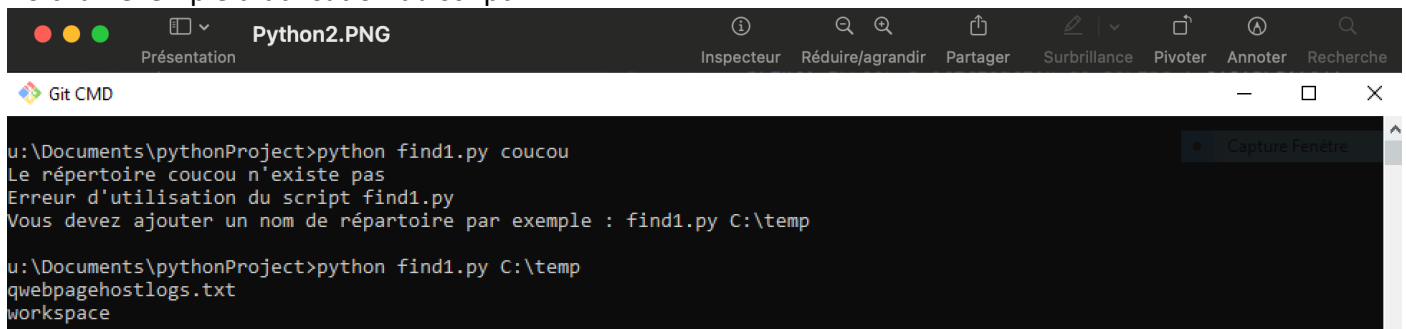
- Vérifie qu'un argument a été proposé (test sur le nombre d'arguments) sinon le script appelle une fonction d'aide
- Vérifie que l'argument est un répertoire qui existe sinon appelle une fonction d'aide
- Une fonction qui permet d'afficher le contenu du répertoire dont le nom est passé en argument en utilisant un itérateur (voir Code 3)

```
def affiche(repertoire):  
    ...
```

*Code 4 Création d'une fonction affiche avec l'argument du répertoire*

- Appeler la fonction d'affichage sur le répertoire passé en argument

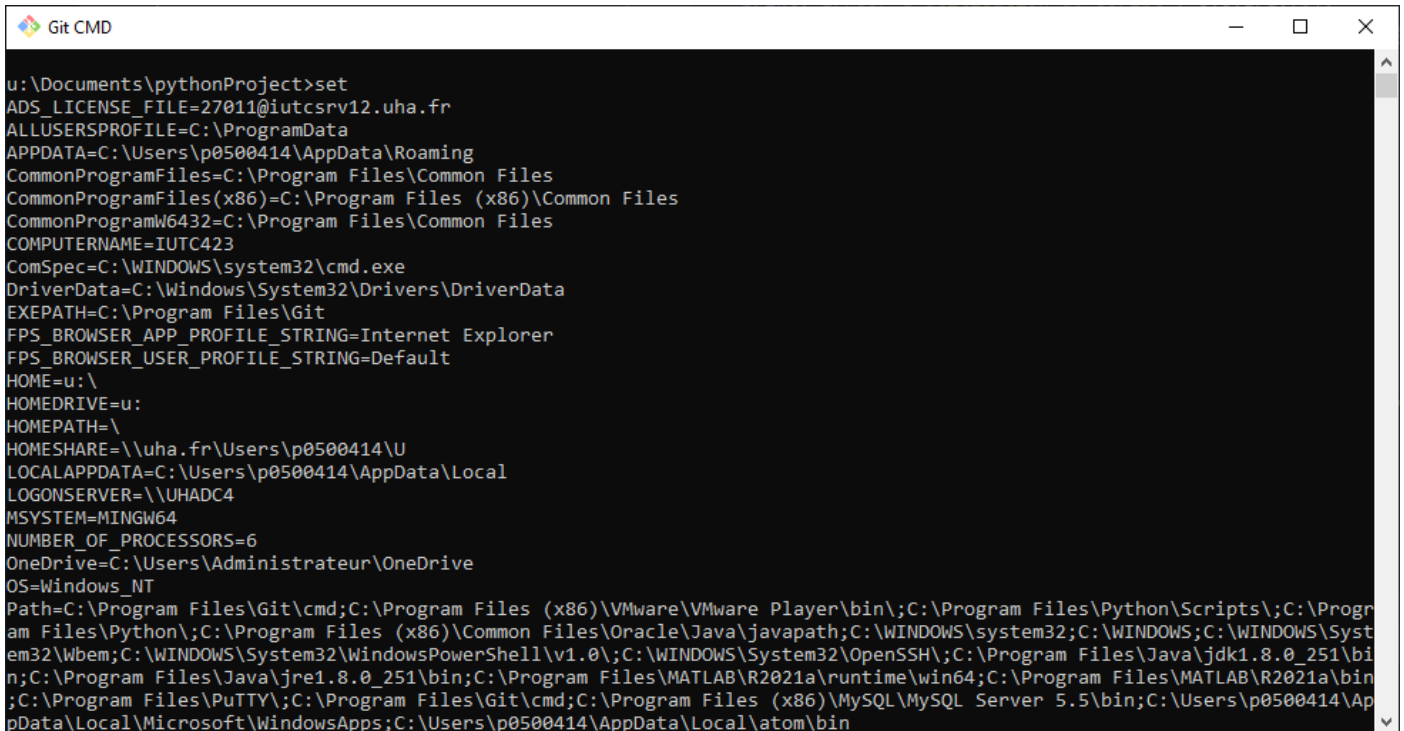
Voici un exemple d'utilisation du script



```
u:\Documents\pythonProject>python find1.py coucou  
Le répertoire coucou n'existe pas  
Erreur d'utilisation du script find1.py  
Vous devez ajouter un nom de répertoire par exemple : find1.py C:\temp  
  
u:\Documents\pythonProject>python find1.py C:\temp  
qwebpagehostlogs.txt  
workspace
```

*Figure 2 Exemple d'utilisation du programme find1.py*

<sup>6</sup> La recherche sur internet va vous proposer d'utiliser la fonction `os.walk()` . Cet usage est à proscrire autrement dit vous n'avez pas le droit de l'utiliser



```
u:\Documents\pythonProject>set
ADS_LICENSE_FILE=27011@iutcsrv12.uha.fr
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\p0500414\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=IUTC423
ComSpec=C:\WINDOWS\system32\cmd.exe
DriverData=C:\Windows\System32\Drivers\DriverData
EXEPATH=C:\Program Files\Git
FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
FPS_BROWSER_USER_PROFILE_STRING=Default
HOME=u:\
HOMEDRIVE=u:
HOMEPATH=\
HOMESHARE=\\uha.fr\Users\p0500414\U
LOCALAPPDATA=C:\Users\p0500414\AppData\Local
LOGONSERVER=\\UHADC4
MSYSTEM=MINGW64
NUMBER_OF_PROCESSORS=6
OneDrive=C:\Users\Administrateur\OneDrive
OS=Windows_NT
Path=C:\Program Files\Git\cmd;C:\Program Files (x86)\VMware\VMware Player\bin\;C:\Program Files\Python\Scripts\;C:\Program Files\Python\;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\Java\jdk1.8.0_251\bin;C:\Program Files\Java\jre1.8.0_251\bin;C:\Program Files\MATLAB\R2021a\runtime\win64;C:\Program Files\MATLAB\R2021a\bin;C:\Program Files\PuTTY\;C:\Program Files\Git\cmd;C:\Program Files (x86)\MySQL\MySQL Server 5.5\bin;C:\Users\p0500414\AppData\Local\Microsoft\WindowsApps;C:\Users\p0500414\AppData\Local\atom\bin
```

Figure 3 Affichage des variables d'environnement Windows (commande set), à noter que vous récupérez une table d'association sous Python

### Exercice 3 : faire une recherche et stocker les répertoires et les fichiers – find2.py

Copiez le script find1.py et modifiez-le pour obtenir la liste des fichiers et des répertoires.

Pour effectuer cette liste, vous devez tester si le contenu est un fichier ou un répertoire ou autres choses<sup>7</sup>. Pour cela, vous avez besoin d'utiliser la commande :

- `os.path.isfile(nom)` : répond vrai si `nom` est un fichier standard.
- `os.path.isdir(nom)` : répond vrai si `nom` est un répertoire.
- Attention : l'utilisation des commandes requiert que le fichier soit directement accessible soit avec un chemin relatif et qu'il existe à partir du répertoire d'exécution soit par un chemin absolu.

Voici les étapes à suivre :

- Renommez votre fonction `affiche` par `recherche`.
- Juste après les imports, créez deux listes vides dont voici les noms : `listeFichiers` et `listeRepertoires`

```
listeFichiers = []
listeRepertoires = []
```

Code 5 Création des listes vides

- Dans la fonction `recherche`, vous allez ajouter dans la `listeRepertoires` les répertoires et dans `listeFichiers` les fichiers. Il suffit pour cela d'utiliser un test simple en utilisant les commandes montrées au début de cet exercice.
  - Il existe de nombreuses fonctions sur les listes que vous pouvez aller voir dans la documentation.
  - Pour ajouter un élément dans une liste, il suffit d'utiliser la commande `append()` :

```
def recherche(repertoire):
    liste = os.listdir(...)
    for ... :
        if ... :
            listeFichiers.append(repertoire + elt)8
            # elt correspond au repertoire à ajouter dans la liste
```

<sup>7</sup> Vous pouvez chercher quels sont les autres types de fichiers.

<sup>8</sup> Il s'agit d'un appel orienté objet, l'objet est la liste et l'appel de la fonction (méthode) est `append`.

```

        # repertoire + elt représente le chemin complet sur le fichier9
    elif ... :
        listeRepertoires.append(repertoire + elt)8
        # elt correspond au fichier à ajouter dans la liste
        # repertoire + elt représente le chemin complet sur le fichier9

```

Code 6 Ajout d'éléments dans une liste

- Pour terminer ce script, nous allons boucler sur la liste des repertoires (`listeRepertoires`) et faire une recherche dans chacun des fichiers.
  - Dans la partie main, une fois le répertoire testé (il doit exister !)
- Ajouter le repertoire dans la liste de repertoires (`listeRepertoires`) en donnant un chemin depuis le repertoire initial
- Ajouter une boucle sur l'ensemble des éléments de `listeRepertoires` et appeler pour chacun d'entre eux la fonction recherche.
- Afficher l'ensemble des fichiers et des repertoires des deux listes.

## Exercice 4 : un find complet – find.py

Recopier le script `find2.py` en `find.py`.

L'idée de ce script est de faire un `find` comme sous un système de type Unix ou `dir /S` sous Windows comme le montre les figures ci-dessous :

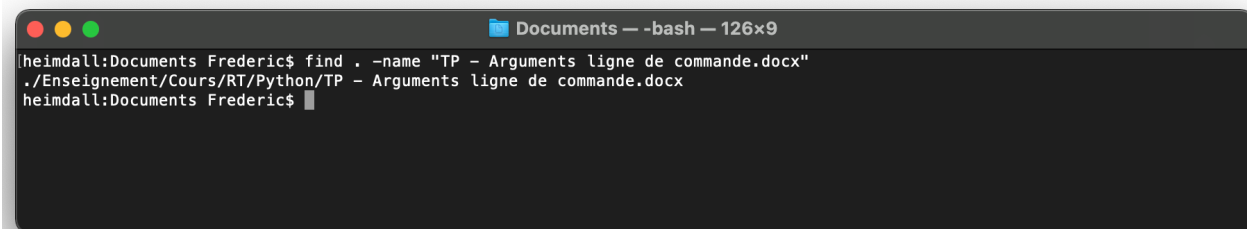


Figure 4 Commande find sous un système de type Unix

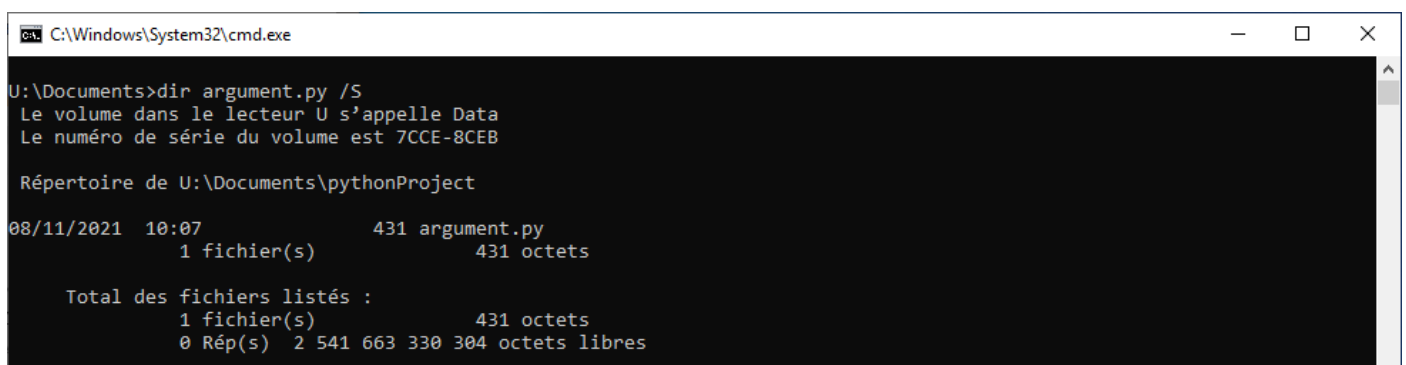


Figure 5 Commande dir /S sur MS Windows

1. Ajouter un nouvel argument en plus du repertoire dans la fonction recherche pour avoir un repertoire et un fichier à rechercher.

```

def recherche(repertoire, fichier):
    ...

```

Code 7 deux arguments dans la fonction recherche

2. Chaque fois que vous allez trouver un fichier, vous allez comparer le nom du fichier trouvé et si les deux noms correspondent, vous ajoutez le fichier, son répertoire dans la liste `listeFichiers`.

<sup>9</sup> Par exemple : vous trouvez un répertoire `workspace` dans `c:\temp`, vous devez donc mettre dans `listeRepertoires` : `c:\temp\workspace`

3. Une fois la recherche effectuée sur chacun des répertoires, vous pouvez afficher la liste des fichiers trouvés (`listeFichiers`) à l'aide d'une boucle `for`.
4. Nous allons maintenant ajouter des arguments de ligne de commandes pour obtenir une commande du type :

```
U:\Documents\PythonProject> python find.py -d C:\temp -f toto.txt
```

*Code 8 Arguments et exemple de lancement du programme find.py*

- Avec `C:\temp` le repertoire initial
- `Toto.txt` : nom du fichier rechercher
- Vous cherchez les fichiers dans le repertoire `C:\temp` et tous ses sous-repertoires les fichiers `toto.txt`
- Pour faire vos tests, je vous conseille de créer une arborescence de tests, par exemple :

```
C:\temp
  toto.txt
  test1
    toto.txt
  test2
    test4
      test6
        toto.txt
  test3
    test5
      toto.txt
```

*Code 9 Arborescence de test*

Le résultat de votre script doit être :

```
U:\Documents\PythonProject> python find.py -d C:\temp -f toto.txt
C:\temp\toto.txt
C:\temp\test1\toto.txt
C:\temp\test2\test4\test6\toto.txt
C:\temp\test3\test5\toto.txt
```

*Code 10 Exemple d'exécution du script sur une arborescence*

Pour terminer vous trouverez sur la page suivante la structure finale de votre script (Code 11).

```

import ...
import ...

listeRepertoires = []
listeFichiers = []

def aide(msg):
    print(msg)
    print(...) # affiche le nom du script et comment il faut appeler ce script
    exit

def recherche(repertoire, fichier):
    # Lister les fichiers du répertoire en vous plaçant dans celui-ci
    contenuDuRépertoire = os.listdir(...)
    for ... : # pour chaque élément (elt) du répertoire
        if ... : # si c'est un répertoire
            listeRepertoires.append(repertoire + "/" + elt)

        elif ... : # sinon si c'est un fichier
            if ... : # si c'est le même de fichier
                listeFichiers.append(repertoire + "/" + elt)

if __name__ == '__main__':
    if ... : # mauvais nombre d'arguments
        aide("Mauvais nombre d'arguments")
    else:
        repertoire = ""
        fichier = ""
        for i in range(1, len(sys.argv)): # je recherche les arguments
            if sys.argv[i] == "-d":
                repertoire = sys.argv[i+1]
            elif sys.argv[i] == "-f":
                fichier = sys.argv[i+1]

        if ... : # si le repertoire n'existe pas
            aide()
        else :
            if ... : # si repertoire ou fichier sont vides
                aide()
            else:
                for dos in listeRepertoires:
                    recherche(dos, fichier)

                for fiche in listeFichiers :
                    print(fiche)

```