

TD1 : Gestion des arguments en CLI et appels systèmes : exemple du find

Les notions abordées

Au cours de ce TD vous allez revoir des éléments de base, aussi aborder notions nouvelles :

- La structure conditionnelle et les boucles
- 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
- L'exécution des scripts dans un IDE et dans une fenêtre d'invites de commandes (cmd ou même Powershell).
- Création et gestion de module / paquet

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 le nom du script :

- `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.

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

- `sys.argv` : liste d'argument qui provient du module `sys`, ainsi `argv[0]` vous donne le nom du script

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

Code 1 Affichage du premier argument avec un main

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

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

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

- Les itérateurs : 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

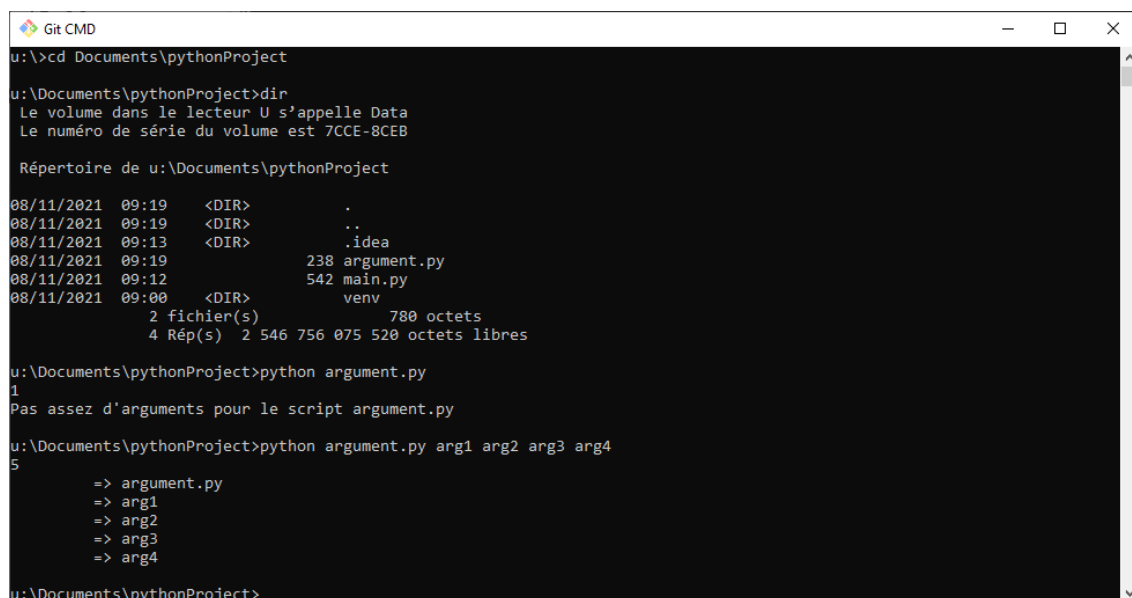
Exercice 1 : argument.py

Coder dans un programme Python les éléments suivants :

- Un environnement d'exécution principal qui affiche le nombre d'arguments en vérifiant leur nombre :
 - Si le nombre est égal à 1 affiche « Pas assez d'arguments pour le script 'nom du script' »
 - Si le nombre est supérieur strictement à 1 affiche les arguments à l'aide d'une boucle.

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

Vous allez maintenant lancer le programme en ligne de commande. Pour cela, vous devez savoir où vous avez enregistré votre script. Dans la capture écran ci-dessous, le script se trouve 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 dossier – find1.py

Il est souvent nécessaire d'accéder au contenu d'un dossier. Vous devez connaître la notion de chemin absolu et de chemin relatif.

Pour exploiter une liste d'un répertoire, vous pouvez utiliser les commandes suivantes :

- `os.chdir(d)` : permet d'aller dans le dossier `d` correspondant
- `os.listdir(d)` : permet d'obtenir une liste de fichiers et dossiers contenu dans le dossier `d`.
- `os.environ[nom_variable]` : liste des variables d'environnement que vous pouvez consulter en tapant dans une fenêtre cmd la commande **set**
 - Par exemple, dans un script python, avec `os.environ["USERNAME"]`, vous obtenez votre nom d'utilisateur (eXXXXXXXX).
- `os.path.exists(d)` : permet de vérifier l'existence du dossier `d`

Proposer le module `find1.py`, qui permet de :

- Vérifier qu'un argument a été proposé (test sur le nombre d'arguments) sinon le script affiche un message d'erreur.
- Vérifier que l'argument est un dossier qui existe sinon affiche un message d'erreur.
- Appeler la fonction 'affiche' qui permet d'afficher le contenu du dossier dont le nom est passé en argument en utilisant un itérateur.
- L'affichage du message d'erreur se fera avec une fonction `aide()` qui précise également comment il faut utiliser le scripte.

L'ensemble de ces instructions constituent le corps de la fonction 'start_find1()' qui est lancée uniquement si le module est appelé en tant que programme principal (`if __name__ == '__main__':`)

```
def affiche(dossier):  
    ...
```

Code 4 Création d'une fonction affiche avec l'argument 'dossier'

- ✚ Pour votre module ainsi pour l'ensemble de ces fonctions, vous rajouterez du **docstring** précisant le rôle du module et des fonctions. Ce docstring doit être vérifié avec la fonction `help` dans le mode interactif.

Voici un exemple d'utilisation du module :

```
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

Exercice 3 : rechercher et stocker les dossiers et les fichiers – find2.py

Proposer le module find2.py qui permet d'obtenir, pour un dossier donné en argument, la liste des fichiers et des dossiers présents dans ce dossier et ses sous dossiers.

Pour avoir cette liste, vous devez tester si le contenu est un fichier ou un répertoire ou autres choses. Pour cela, vous avez besoin d'utiliser la commande :

- `os.path.isfile(nom)` : renvoie vrai si 'nom' est un fichier standard.
- `os.path.isdir(nom)` : renvoie 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 :

- Définir une fonction `recherche` qui prend en argument le nom d'un dossier et renvoie les deux listes : `listeFichiers` et `listeDossiers`
- Dans la fonction `recherche`, vous allez ajouter dans la `listeDossiers` les dossiers et dans `listeFichiers` les fichiers. Il suffit pour cela d'utiliser un test simple en utilisant les commandes montrées précédemment.
- Pour terminer ce script, il faut faire une boucle sur la liste des dossiers obtenus (`listeDossiers`) afin d'ajouter les sous dossiers dans cette liste, idem pour les fichiers.
- Afficher l'ensemble des fichiers et des répertoires des deux listes.

L'ensemble de ces instructions constituent le corps de la fonction `'start_find2()'` qui est lancée uniquement si le module est appelé en tant que programme principal (`if __name__ == '__main__':`)

⚠ Pour votre module ainsi pour l'ensemble de ces fonctions, vous rajouterez du **docstring** précisant le rôle du module et des fonctions. Ce docstring doit être vérifié avec la fonction `help` dans le mode interactif.

Voici un squelette du code à compléter :

```
def recherche(dossier):
    liste = os.listdir(...)
    for ... :
        if ... :
            listeFichiers.append(dossier + elt)1
            # elt correspond au dossier à ajouter dans la
liste
            # 'dossier + elt' représente le chemin complet
sur le fichier
        elif ... :
            listeDossiers.append(dossier + elt)
```

```

# 'elt' correspond au fichier à ajouter dans
la liste
# 'dossier + elt' représente le chemin complet
sur le fichier

```

Code 5 Ajout d'éléments dans une liste

Exercice 4 : un find complet – find3.py

En se basant sur vos scripts précédents, l'idée maintenant est de faire un `find` comme sous un système de type Unix ou `dir /S` sous Windows comme le montre la figure ci-dessous :

```

C:\Windows\System32\cmd.exe
U:\Documents>dir argument.py /S
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  10:07                431 argument.py
               1 fichier(s)                431 octets

Total des fichiers listés :
               1 fichier(s)                431 octets
0 Rép(s)  2 541 663 330 304 octets libres

```

Figure 4 Commande dir /S sur MS Windows

Proposer le module `find3.py` qui assure les points suivants :

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

```

def recherche(dossier, fichier):
    ...

```

Code 6 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 ajouter le fichier avec son chemin relatif dans la liste `listeFichiers`. Chaque fois que vous allez trouver un dossier, vous allez l'ajouter avec son chemin relatif dans la liste `listeDossiers`.
3. Une fois la recherche effectuée sur chacun des dossiers, vous pouvez afficher la liste des fichiers trouvés (`listeFichiers`) à l'aide d'une boucle `for`.
4. Vous allez 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 7 Arguments et exemple de lancement du programme find.py

- Avec `C:\temp` le dossier initial.
- `Toto.txt` : nom du fichier à rechercher.

- La recherche s'effectue dans le répertoire `C:\temp` et tous ses sous-dossiers.
- Pour faire vos tests, vous pouvez créer une arborescence de tests, par exemple :

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

Code 8 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 9 Exemple d'exécution du script sur une arborescence

L'ensemble de ces instructions constituent le corps de la fonction 'start_find3()' qui est lancée uniquement si le module est appelé en tant que programme principal (`if __name__ == '__main__':`)

Exercice 5 : menu

À présent vous allez proposer un script menu.py qui va permettre à l'utilisateur de choisir un module à appliquer (find1, find2 ou find3) et de lancer la méthode start_findx() correspondante.

Voici un exemple d'utilisation :

```
Menu:
1. find1
2. find2
3. find3
4. find4
5. Quitter
Veuillez choisir une option (1-5):
```

Exercice 6 : my_package

Partie 1 : création du paquet

- 1- À présent vous allez transformer le module find3.py en package. Pour cela, créer le répertoire **my_package_NM** avec NM vos initiales. Dans ce dossier vous devez avoir le fichier `__init__.py` dans lequel vous allez définir la version du package ainsi qu'un message d'accueil qui sera affiché lorsque le package est importé.
- 2- Déplacer le fichier find3.py dans le dossier **my_package_NM**.
- 3- Maintenant, modifier le fichier menu.py afin de faire appel de find3.py à partir de votre paquet.
- 4- Lancer votre scripte menu.py et vérifier bien que le message d'accueil s'affiche bien.

Partie2 : installation du paquet

Dans l'état actuel des choses, votre module find3.py n'est pas reconnu en dehors de son répertoire. À présent vous allez voir comment installer un paquet :

- 1- Dans le répertoire de votre projet créer le fichier setup.py avec les éléments suivants :

```
from setuptools import setup

setup(
    name='my_paquet_IB',
    version='0.1',
    description='un paquet pour gérer des bibliothèques',
    packages=[], # répertoire dans lequel se trouve le paquet
    scripts=["my_paquet_IB\\find4.py"] #module contenu dans le paquet
)
```

- 2- Lancez le script d'installation du paquet avec la commande :

```
python setup.py install --user
```

L'option --user permet d'installer le paquet dans votre répertoire personnel. Si l'option n'est pas spécifiée, python tentera d'installer le paquet dans un répertoire système ce qui provoquera une erreur étant donné que vous n'avez pas les droits nécessaires.

- 3- Déplacez-vous dans un répertoire quelconque, puis entrez la commande `find3.py -d test3 -f toto.txt`

Est-ce que la commande se lance bien ? Qu'est-ce qu'il faut ajouter au niveau de votre code pour afficher le résultat ?