La Gestion des scripts Python

SCRIPTING SHELL ET PYTHON

Enseignant : TOSSOU Kodjo Elom Francis K.

1 Introduction

Python est un langage de programmation très dynamique et offrant beaucoup de possibilité comparativement au shell grâce à son aspect modulaire. Dans un premier il faudra l'installer. Dans ce cours nous utiliserons la version 3 de python dans un environnement linux.

2 Création d'un script Python

Afin de créer un script **python** nous suivrons un certain nombre d'étape :

- **Etape 1 :** Créer un fichier à l'aide d'un éditeur de texte avec l'extension « **.py** ». L'extension ne permet pas au système de savoir quel programme utilisé mais pour nous utilisateur ou administrateur, nous pouvons savoir immédiatement que c'est un script **python**.
- **Etape 2 :** La première instruction à mettre dans le fichier est le « **#!/usr/bin/env python3** » où python3 est la version de python installée sur le système. Cette instruction permet au système de savoir quel programme exécutera les instructions de ce fichier. Cette instruction s'appelle le « **shebang** ».
- **Etape 3 :** La troisième étape consiste à écrire les instructions que l'on souhaite faire exécuter par le script.

Voici donc un exemple de fichier de script Shell :

```
#!/usr/bin/env python3
print("coucou")
```

Nous le nommerons « **coucou.py** » et pour l'exécuter dans le terminal, nous utiliserons la commande suivante :

```
$ coucou.py
```

3 Les opérateurs

Dans le script **python** il existe plusieurs types d'opérateurs.

3.1 Les opérateurs de comparaison

Les opérateurs de comparaison permettent de vérifier une condition et de retourner vrai ou faux. Ces opérateurs sont les suivants :

- < : permet de vérifier si une valeur est inférieure à l'autre
- > : permet de vérifier si une valeur est supérieure à l'autre
- <= : permet de vérifier si une valeur est inférieure ou égale à l'autre
- >= : permet de vérifier si une valeur est supérieure ou égale à l'autre
- != : permet de vérifier si une valeur est différente de l'autre
- == : permet de vérifier si une valeur est égale à l'autre

3.2 Les opérateurs logiques

Les opérateurs logiques permettent de tester deux conditions et de retourner vrai ou faux. Ces opérateurs sont les suivants :

- **and** : retourne vrai uniquement si les deux conditions sont vraies. *Ex : if 1 < 2 and 4 > 3*
- **or** : retourne vrai si l'une des deux conditions est vraie. *Ex : if 1 == 2 or 3 < 4*
- **not** : retourne l'état contraire d'une condition. C'est-à-dire, retourne vrai si la condition est fausse et retourne faux si la condition est vraie. *Ex : if not 1 > 2*

3.3 Les opérateurs numériques

Les opérateurs numériques permettent d'effectuer des opérations sur des chiffres. Ces opérateurs sont les suivants :

- + : permet de faire l'addition de deux nombres. *NB : Placé entre deux chaînes de caractère, il fait la concaténation de ces deux chaînes.*
- - : permet de faire la soustraction de deux nombres.
- * : permet de faire la multiplication de deux nombres.
- / : permet de faire la division de deux nombres.
- ** : permet d'élever en exposant le nombre placé à droite. *Ex : a**b équivaut à a^b*
- % : permet de calculer le modulo, c'est-à-dire le reste de la division de deux nombres.
- // : permet de faire la division de deux nombres et d'arrondir le résultat.

3.4 Les opérateurs d'appartenances

Les opérateurs d'appartenance permettent de vérifier si un objet appartient à un ensemble ou non. Ces opérateurs sont les suivants :

- **in** : retourne **vrai** si l'objet appartient à l'ensemble. *Ex : if 'orange' **in** fruits. Avec **fruits** étant une liste de chaîne de caractères.*
- **not in** : retourne **vrai** si l'objet n'appartient pas à l'ensemble.

4 Exemples Pratiques

4.1 Hello world

Le programme « *Helloworld* » est le tout premier programme qu'un programmeur écrit tout en apprenant un nouveau langage. C'est un programme qui imprime le « *Bonjour le monde* » en tant que sortie. Vous pouvez donc créer un fichier ***helloword.sh*** à l'aide de l'éditeur (*vim* ou *nano*) :

```
$ nano helloworld.py
```

Copiez maintenant les lignes ci-dessous dans « ***helloworld.sh*** » et sauvegardez-les.

```
#!/usr/bin/env python3
Print("Bonjour le monde")
```

Vous pouvez maintenant exécuter la commande :

```
$ python3 helloworld.py
```

Une autre méthode est d'abord de rendre le fichier exécutable :

```
$ chmod u+x helloworld.py
```

Et maintenant, exécutez le fichier en utilisant la commande ci-dessous.

```
$ ./helloworld.py
```

4.2 Utilisation de la fonction « print() »

La fonction ***print()*** est utilisée pour imprimer du texte ou une sortie. Il prend en paramètre des chaînes de caractère et des variables séparés par des virgules.

Syntaxe :

```
print([chaîne de caractère],[variables])
```

NB : Les chaînes de caractères sont délimitées par les ‘’ ‘’

Exemple : Créez un nouveau fichier **print.py** et ajoutez les lignes ci-dessous.

```
#!/usr/bin/env python3
print("Script Scripting Shell et Python")
var = "Je suis un adminsys"
print("Script Scripting Shell et Python", var)
```

Exécutez le fichier avec la commande :

```
$ ./print.py
```

4.3 Utilisation des commentaires

Les commentaires sont des remarques d'un programmeur sur le but ou la raison du code ou de la logique. Il est recommandé d’ajouter des commentaires pour qu’à l’avenir, tout le monde puisse comprendre le code en lisant simplement les commentaires. Les commentaires font partie du code mais sont ignorés par le compilateur. Dans le script **python**, toute ligne commençant par « # » est considérée comme un commentaire. Par exemple :

```
#!/usr/bin/env python3
# Ceci est un commentaire
print("Je suis un commentaire")
```

Ici « **# Ceci est un commentaire** » est un commentaire, et quand nous exécuterons ce script, le l’interpréteur ignorera la ligne.

Les commentaires peuvent être des :

- Commentaires sur une seule ligne : Dans ce cas, il faut utiliser « # »
- Commentaires sur plusieurs lignes : Dans ce cas, il faut utiliser « """ commentaires """ »

Exemple de commentaire sur plusieurs lignes :

```
#!/usr/bin/env python3
"""
Ce script calcul
La somme de 2 et 8.
"""
sum = 2+8
# le résultat sera
print("la somme est : ", sum)
```

Enregistrez le fichier sous ***multicommentaires.py*** et exécutez-le à partir de la commande :

```
$ python3 multicommentaires.py
```

4.4 Utilisation de variables

Les variables sont des symboles nommés utilisés pour stocker temporairement des valeurs. Il peut s'agir d'une chaîne ou d'une valeur numérique que nous pouvons utiliser à n'importe quel endroit du script. Vous pouvez créer des variables et leur attribuer des valeurs. Les noms de variable doivent être descriptifs afin que vous puissiez comprendre l'objectif pour lequel vous avez créé cette variable. En python le typage des variables est dynamique.

Nous avons plusieurs types de variables en python :

- **integer** : Il permet de contenir les nombres entier. *Exemple : 10*
- **float** : Il permet de stocker les entiers et les nombres à virgule. *Exemple : 3.14*
- **boolean** : Il permet de stocker deux types de valeurs : **True** pour l'état vrai et **False** pour l'état faux.
- **string** : Il permet de stocker les chaînes de caractères. *Exemple : 'abalo'*
- **list** : Il permet de stocker des listes de données qui peuvent être de n'importe quel type. *Exemple : [1, 2, 'tata, 'titi'].*
- **tuple** : Il permet de stocker des données qui ne pourront plus être modifiées. *Exemple : ('abalo', 2, 'tata').*
- **set** : Il permet de stocker un ensemble de données. Mais il ne contient qu'une seule instance de chaque donnée. Ce qui veux dire qu'on ne peut avoir deux fois la même valeur. *Exemple : {'a', 'b', 'c', 'd'}*
- **dictionary** : Il permet de stocker les données sous forme de clé et valeur. Il devient alors aisé de retrouver une valeur à partir de sa clé. *Exemple : {'a':1, 'b':2}*

Les variables définies par l'utilisateur sont celles que nous définissons dans notre script. Par exemple, nous avons la variable « annee » pour stocker l'année en cours comme ci-dessous :

```
annee = 2022
```

Et nous pourrons l'utiliser plus tard :

```
print(annee)
```

Vous pouvez voir que nous avons utilisé $ pour référencer sa valeur.

Exemple : Créez un fichier **variables.py** et ajoutez-y les lignes ci-dessous.

```
#!/usr/bin/env python3
website = 'www.esgis.tg'
annee = 2022

print("Bienvenue sur " + website)
print("Annee = ", annee)
```

Enregistrez-le et testez-le.

4.5 Obtenir l'entrée de l'utilisateur

Obtenir la contribution de l'utilisateur est très crucial pour rendre un script interactif, donc à cette fin dans le script **python**, nous utilisons la fonction **input**.

```
#!/usr/bin/env python3
nom = input("Veuillez saisir votre nom : ")
print("Votre nom est " + nom)
```

4.6 Utilisation d'arguments en ligne de commande

Nous pouvons également lire l'entrée de l'utilisateur à partir d'arguments de commande, tout comme n'importe quel autre langage de programmation. Nous pouvons ensuite utiliser ces arguments dans nos scripts comme, selon le nombre d'arguments que nous avons fournis.

Pour pouvoir interagir avec le terminal et récupérer les arguments, il nous faut utiliser des fonctions spéciales. Ces fonctions se trouvent dans un module. Pour importer des modules, cela se fait avec les instructions suivantes : *NB **[module]** est le nom du module et **[function]** est le nom de la fonction.*

```
from [module] import [function]
```

ou

```
from [module] import [function] as [f]
```

ou

```
import [module]
```

Créez un fichier ***arguments.py*** et copiez les lignes ci-dessous à l'intérieur. Ici nous importons le module **sys** afin de pouvoir utiliser la fonction **argv** qui est utile pour récupérer les arguments.

```
#!/usr/bin/env python3

import sys

print("Nombre d'arguments : " + len(sys.argv))
print("Nom : " + str(sys.argv[1]))
print("Contact : " + str(sys.argv[2]))
```

Exécutez maintenant le fichier script **arguments.py** avec deux paramètres supplémentaires après son nom comme ici :

```
$ python3 arguments.py TOSSOU 90000000
```

4.7 Utilisation de boucles

Les boucles sont utilisées dans tous les langages de programmation où vous devez exécuter le même code de façon répétitive. Il existe deux types de boucles (**while** et **for**). Nous verrons chacun.

4.7.1 Boucle « while »

Il est utilisé lorsque vous avez besoin de répéter la ligne de code un nombre inconnu de fois jusqu'à ce qu'il satisfasse à certaines conditions. L'indentation est très importante. Les instructions de la boucle doivent être décalée vers la droite par rapport à la boucle afin de se retrouver à l'intérieur de la boucle. Voici sa structure :

```
#!/usr/bin/env python3
while [CONDITION]:
  [INSTRUCTIONS]
```

La condition est évaluée avant d'exécuter les commandes à chaque itération, et elle continuera à s'exécuter jusqu'à ce que la condition évalue à faux, et la boucle sera terminée.

```
#!/usr/bin/env python3
i = 0
while i <= 4 :
  print("Numéro : ", i)
  i++
```

4.7.2 Boucle « for »

La boucle ***for*** s'itère sur une liste d'éléments et exécute l'ensemble donné de commandes. La boucle **for** dans python prend la forme suivante :

```
#!/usr/bin/env python3
for item in [LIST]:
  [COMMANDS]
```

Dans l'exemple ci-dessous, la boucle s'itérera sur chaque élément et générera une table de variable **i**.

```
#!/usr/bin/env python3
i = 2
for compteur in range(10) :
      resultat = i * compteur
      print(i," x ",compteur, " = ",resultat)
```

4.8 Utilisation de déclarations conditionnelles

Les énoncés conditionnels sont l'un des concepts fondamentaux de tout langage de programmation. Vous prenez des décisions en fonction de certaines conditions remplies.

4.8.1 Déclaration « if »

Dans un script python, la condition « if » a plusieurs formes, mais regardons l'état de base. Vous pouvez l'utiliser la condition « if » avec des conditions simples ou multiples.

```
if Condition :
  INSTRUCTIONS
```

Créez un fichier nommé **simple_if.py** et ajoutez-y ce code :

```
#!/usr/bin/env python3

nombre = input("Saisissez un nombre : ")

if nombre > 10 :
  print("Le nombre est plus grande que 10.")
```

Il est demandé à l'utilisateur de saisir un nombre. Si le nombre est plus grand que **10** alors la sortie « Le nombre est plus grande que 10. » sera affichée. Autrement, rien ne sera affiché.

4.8.2 Déclaration « if else »

Maintenant, nous allons ajouter le bloc « if else », qui s'exécutera si la condition est fausse.

```
if Condition :
  STATEMENTS1
else:
  STATEMENTS2
```

Nous allons donc modifier l'exemple ci-dessus comme suit :

```
#!/usr/bin/env python3

nombre = input("Sasissez un nombre: ")

if nombre > 10 :
  print("La variable est plus grande que 10.")
else :
  print("La variable est égale ou plus petite que 10.")
```

Si vous exécutez le code et entrez un nombre, le script affiche une chaîne de caractère en fonction du fait que le nombre est supérieur ou inférieur/égal à 10.

4.8.3 Déclaration « if elif »

Python a une syntaxe équivalente pour **sinon si** ainsi :

```
if Condition1 :
  STATEMENTS1
elif Condition2 :
  STATEMENTS2
else :
  STATEMENTS3
```

Donc, après avoir modifié l'exemple ci-dessus comme suit :

```
#!/usr/bin/env python3

nombre = int(input("Saisissez un nombre: "))

if nombre > 10 :
  print("La variable est plus grande que 10.")
elif nombre == 10 :
  print("La variable est égale à 10.")
else :
  print("La variable est plus petite que 10.")
```

Nous utilisons `int()` pour convertir la saisie de l'utilisateur en entier, sinon il le prendra comme une chaîne de caractère.

4.9 Utilisation des fonctions

Tout comme d'autres langages de programmation, le script **python** a également le concept de fonctions. Il permet à l'utilisateur d'écrire un bloc de code personnalisé qui sera nécessaire pour être réutilisé encore et encore.

```
Syntaxe :

def NomFonction():
  instructions
```

Maintenant, nous allons créer une fonction **somme** qui prendra les numéros d'entrée de l'utilisateur et affichera la somme de ces nombres comme sortie.

```
#!/usr/bin/env python3
def somme() :
  a = int(input("Entrez le premier chiffre : "))
  b = int(input("Entrez le second chiffre : "))
  print("La somme est : ", a+b)
}

Somme()
```

4.10 Créer une fonction avec paramètres

Nous pouvons également créer des fonctions avec paramètres afin d'effectuer des actions sur des variables définies en dehors de la fonction.

Pour mieux comprendre, nous allons créer un fichier nommé **fonction_parametre.py** et ajouter le code suivant.

```
#!/usr/bin/env python3

def Rectangle_Area(longueur, largeur) :
      area = longueur * largeur
      print("L'aire est : ", area)

Rectangle_Area(10,20)
```

NB : Ici, la fonction ***Rectangle_Area(longueur, largeur)*** calculera l'aire d'un rectangle en fonction des valeurs de paramètres.

4.11 Passer la valeur de retour d'une fonction

La fonction python peut passer à la fois les valeurs numériques et les chaînes de caractères. Elle le fait grâce à l'instruction **return**. Pour bien comprendre nous allons créer un fichier nommé **fonction_retour.py** et y ajoutez le code suivant.

```
#!/usr/bin/env python3
def salutation(nom) :
      salut = "Bonjour, " + nom
      return salut

nom = input("Saisissez votre nom : ")
val = salutation(nom)
print("La valeur de retour de la fonction est ",val)
```

NB : La fonction, ***salutation(nom)*** renvoie une valeur de chaîne dans la variable, **val** qui imprime plus tard en combinant avec d'autres cordes.

4.12 Vérifiez le nombre pair/impair

Dans notre prochain exemple, nous allons écrire un script python qui acceptera un numéro d'entrée de l'utilisateur et affichera si le nombre donné est un nombre pair ou impair.

```
#!/usr/bin/env python3

n = int(input("Veuillez saisir un nombre : "))
reste = n%2
if reste == 0 :
      print("C'est un nombre pair")
else :
      print("C'est un nombre impair")
```

4.13 Création de répertoires

L'exemple suivant vous montrera comment créer un répertoire à partir d'un script python. Le script recevra le nom du dossier et vérifiera s'il existe déjà ou non. Dans le cas où il existe, Il affichera comme message « ***Le répertoire existe déjà*** » sinon, il créera un répertoire. Pour cela nous importerons le module **os** qui contient des fonctions qui permettent d'interagir avec le système d'exploitation. La fonction **system** nous permet de pouvoir exécuter des commandes dans le terminal.

```
#!/usr/bin/env python3

import os

dir = input("Veuillez entrer le nom d'un dossier : ")
if os.path.exists(dir):
      print("Le répertoire existe déjà !")
else :
      os.system("mkdir " + dir)
      print("Le répertoire a bien été créé !")
```

4.14 Tester s'il existe un fichier

Il est aussi possible de vérifier l'existence d'un fichier en python en utilisant la fonction **path.isfile** ou **path.exists**. La fonction **path.exists** renvoie vrai même s'il s'agit d'un dossier et non d'un fichier. Pour cette raison, il est préférable d'utiliser **path.isfile** pour tester si un fichier existe. Créez un fichier nommé « ***fichier_existe.py*** » et ajoutez le code suivant :

NB : Ici, le nom de fichier passera de la ligne de commande. Nous utiliserons le module **sys** afin de récupérer les arguments.

```
#!/usr/bin/env python3

import os, sys

nom_de_fichier = str(sys.argv[1])

if os.path.isfile(nom_de_fichier) :
      print("Le fichier existe !")
else :
      print("Le fichier n'existe pas !")
```

4.15 Lecture de fichiers

En utilisant Python, vous pouvez lire des fichiers très efficacement. L'exemple ci-dessous montre comment lire un fichier à l'aide de scripts python. Créez un fichier appelé « **entreprises.txt** » avec le contenu suivant.

```
Google
Amazone
Microsoft
Macdonald
KFC
Apple
```

Ce script lira le fichier ci-dessus et affichera la sortie. La fonction **open()** nous permet d'ouvrir un fichier selon différent mode :

- **r** pour **read**. Il s'agit de l'ouvrir en mode lecture
- **w** pour **write**. Il s'agit de l'ouvrir en mode écriture. Si le fichier n'existe pas, ce mode permet de le créer. Si le fichier existe déjà, il écrasera le contenu du fichier ;
- **w+** pour l'ouvrir en écriture et lecture en même temps ;
- **a** pour **append**. Il s'agit de l'ouvrir en mode ajout. La grande différence avec le mode écriture est que si le fichier contient déjà des données, il ne les écrase pas mais y ajoute les nouvelles informations ;
- **a+** pour l'ouvrir en ajout et en lecture simultanément.

Pour fermer un fichier, la fonction **close()** est utilisée. La syntaxe de la fonction open est la suivante : `open('chemin vers fichier', 'mode').`

```
#!/usr/bin/env python3
file = open('entreprises.txt','r')
line = file.readline()
while line :
      print(line)
      line = file.readline()
file.close()
```

4.16 Ajouter du contenu à un fichier

De nouvelles données peuvent être ajoutées dans n'importe quel fichier existant en utilisant le mode '**a**' dans la fonction **open**. Créez un fichier nommé « **ajout_fichier.py** »et ajoutez le code suivant pour ajouter du nouveau contenu à la fin du fichier. Ici, « ***Apprentissage de script python*** » sera ajouté au fichier « **SSP.txt** » après l'exécution du script :

```
#!/usr/bin/env python3

import os

print("Avant l'ajout au fichier")
os.system("cat SSP.txt")
file = open('SSP.txt','a')
file.write("Apprentissage de script python")
print("Après l'ajout au fichier")
os.system("cat SSP.txt")
```

5 Conclusion

Les scripts python peuvent être utiles. Des tâches complexes exécutées de manière appropriée peuvent augmenter votre productivité dans une large mesure et vous aider à résoudre les problèmes en un rien de temps. De plus, il n'y a pas de limite à son évolutivité. En tant qu'administrateur système Linux, je vous recommande fortement de maîtriser les exemples vus dans ce cours.