Travaux pratiques - Revue de programmation Python

# Objectifs

Partie 1 : Lancer la DEVASC VM

Partie 2 : Démarrer le code Python et VS

Partie 3 : Examiner les types de données et les variables

Partie 4 : Listes de révision et dictionnaires

Partie 5 : Revue de la fonction de saisie

Partie 6 : Revue des fonctions If, For, et While

Partie 7 : Examiner les méthodes d'accès aux dossiers

# Contexte/scénario

Dans ce TP, vous passez en revue les compétences de base en programmation Python, y compris les types de données, les variables, les listes, les dictionnaires, les entrées utilisateur, les instructions if, les boucles for et while et l'accès aux fichiers. Ce TP n'est pas destiné à remplacer une expérience de programmation antérieure et ne couvre pas nécessairement toutes les compétences Python dont vous aurez besoin pour ce cours. Cependant, ce TP devrait servir de bonne mesure de vos compétences en programmation Python et vous aider à vous diriger vers l'endroit où vous pourriez avoir besoin de plus d'examen.

**Note** : Ceci est un rappel. Assurez-vous d'observer les conventions d'indention Python correctes lors de l'écriture de vos scripts. Si vous avez besoin d'un tutoriel, recherchez « Règles d'indention Python » sur Internet.

# Ressources requises

* 1 PC avec système d'exploitation de votre choix
* Boîte virtuelle ou VMWare
* Machine virtuelle DEVASC

# Instructions

## Lancez la machine virtuelle DEVASC

Si vous n'avez pas encore terminé **TP - Installer l'environnement de laboratoire de machines virtuelles**, faites-le maintenant. Si vous avez déjà terminé ce laboratoire, lancez la **machine virtuelle DEVASC**.

## Démarrage du code Python et VS

Dans cette partie, vous allez passer en revue le démarrage de l'interpréteur interactif de Python et l'utilisation de Visual Studio Code pour écrire et exécuter un script « Hello World ».

### Démarrez Python.

* + - 1. Pour vérifier la version Python en cours d'exécution dans la machine virtuelle, ouvrez une fenêtre de terminal et entrez la commande **python3 -V**. C'est une bonne commande à exécuter si vous deviez installer Python sur un autre ordinateur ou si vous avez besoin de vérifier quelle version est installée.

devasc@labvm:~$ **python3 -V**

Python 3.8.2

**Remarque** : Vous devez le changer en **python2 -V** si un autre périphérique que vous utilisez exécute la version 2. Cependant, depuis le 1er janvier 2020, Python 2 n'est plus pris en charge. Par conséquent, Python 2 n'est pas pris en charge dans ce laboratoire ou ce cours.

**Remarque** : Au moment de la rédaction de ce labo, Python 3.8.2 était la dernière version. Bien que vous puissiez mettre à jour votre installation Python avec la commande **sudo apt-get install python3**, ce laboratoire et le reste des laboratoires de ce cours sont basés sur Python 3.8.2.

* + - 1. Pour démarrer Python, tapez **python3**. Les trois crochets (>>>) indiquent que vous êtes dans l'interpréteur interactif de Python.

devasc @labvm ~$ **python3**

Python 3.8.2 (default, Mar 13 2020, 10:14:16)

[GCC 9.3.0] on linux

Tapez "help", "copyright", "credits" ou "license" pour plus d'informations.

>>>

### Utilisez l'interprète comme une calculatrice.

* + - 1. De là, vous pouvez effectuer une variété de tâches de programmation de base, y compris des opérations mathématiques. Le tableau présente la syntaxe Python à utiliser pour les opérations mathématiques les plus courantes.

| Opération | Maths | Syntaxe |
| --- | --- | --- |
| Addition | a+b | a+b |
| Soustraction | a-b | a-b |
| Multiplication | axb | a\*b |
| Division | a÷b | a/b |
| Exposants | a^b | a\*\*b |

Entrez quelques opérations mathématiques à l'aide de la syntaxe Python, comme illustré dans les exemples.

>>> **2+3**

5

>>> **10-4**

6

>>> **2\*4**

8

>>> **20/5**

4,0

>>> **3\*\*2**

9

* + - 1. Rappelons que Python utilise l'ordre standard des opérations communément appelé PEMDAS. Les expressions mathématiques sont évaluées dans l'ordre suivant.

**P**arenthèses

**E**xposants

**M**ultiplication et **D**ivision

**A**ddition et **S**oustraction

Essayez de saisir une expression avec un ordre complexe d'opérations dans l'interpréteur interactif.

### Utilisez l'interpréteur interactif pour imprimer une chaîne.

Une chaîne est une séquence de caractères telle que des lettres, des chiffres, des symboles ou des signes de ponctuation. L'interpréteur interactif affiche directement le texte que vous entrez sous forme de chaîne tant que vous placez la chaîne entre guillemets simples (') ou doubles guillemets (").

* + - 1. Tapez “Hello World!” ou ‘Hello World!’ dans l'interpréteur interactif.

>>> **"Hello World!"**

'Hello World!'

>>> **'Hello World!'**

'Hello World!'

* + - 1. La commande **print** peut également être utilisée directement dans l'interpréteur interactif.

>>> **print("Hello World!")**

Hello World!

* + - 1. Pour quitter l'interpréteur interactif, entrez **quit ()**.

>>> **quit()**

devasc@labvm:~$

### Ouvrez VS Code et créez un script pour Hello World.

De nombreux environnements de développement sont disponibles pour les programmeurs pour gérer leurs projets de codage. Dans ce cours, les laboratoires utiliseront l'installation de Microsoft Visual Studio Code (VS Code) par la machine virtuelle.

* + - 1. Ouvrez le code VS. Si c'est votre première fois, une fenêtre de **bienvenue** vous sera probablement présentée.
      2. N'hésitez pas à explorer les menus et options VS Code sur votre temps libre. Pour l'instant, cliquez sur **File > New File** pour ouvrir un nouveau fichier.
      3. Dans votre nouveau fichier, tapez la commande **print** de l'étape précédente.
      4. Enregistrez le script sous **hello-world.py** dans le dossier **labs/devnet-src/python**. Assurez-vous d'ajouter l'extension **.py** pour le fichier Python.
      5. Pour exécuter le script, cliquez sur **Run > Run Without Debugging**. Une fenêtre de terminal s'ouvre dans VS Code, exécute le code pour lancer une instance de Python, exécute votre script, puis quitte Python à votre ligne de commande Linux.

devasc@labvm:~/labs/devnet-src/python$ env DEBUGPY\_LAUNCHER\_PORT=36095 /usr/bin/python3 /home/devasc/.vscode/extensions/ms-python.python-2020.4.76186/pythonFiles/lib/python/debugpy/no\_wheels/debugpy/launcher /home/devasc/labs/devnet-src/python/hello-world.py

Hello World!

devasc@labvm:~/labs/devnet-src/python$

* + - 1. Maintenant que vous avez une ligne de commande ouverte dans VS Code, vous pouvez lancer manuellement Python et exécuter rapidement votre script avec la commande suivante.

devasc@labvm:~/labs/devnet-src/python$ **python3 hello-world.py**

Hello World!

devasc@labvm:~/labs/devnet-src/python$

* + - 1. Vous pouvez également ouvrir une fenêtre de terminal en dehors de VS Code et entrer la même commande en vous assurant de fournir des informations de chemin d'accès.

devasc@labvm:~$ **python3 ~/labs/devnet-src/python/hello-world.py**

Hello World!

devasc@labvm:~$

Dans ce cours, vous exécuterez généralement vos scripts directement dans VS Code.

## Examen des types de données et des variables

Dans cette partie, vous utiliserez l'interpréteur interactif pour passer en revue les types de données, créer des variables, concaténer des chaînes et passer entre les types de données.

### Utilisez l'interpréteur interactif pour consulter les types de données de base.

En programmation, les types de données sont une classification qui indique à l'interpréteur comment le programmeur a l'intention d'utiliser les données. Par exemple, l'interpréteur doit savoir si les données saisies par le programmeur sont un nombre ou une chaîne. Bien qu'il existe plusieurs types de données différents, nous nous concentrerons uniquement sur les éléments suivants :

* **Integer** - utilisé pour spécifier des nombres entiers (pas de décimales), tels que 1, 2, 3, et ainsi de suite. Si un entier est entré avec une décimale, l'interpréteur ignore la décimale. Par exemple, 3.75 est interprété comme 3.
* **Float** - utilisé pour spécifier des nombres qui nécessitent une valeur décimale, telle que 3.14159.
* **String** - toute séquence de caractères tels que des lettres, des chiffres, des symboles ou des signes de ponctuation.
* **Boolean** - tout type de données ayant une valeur True ou False.

Utilisez la commande **type()** pour déterminer les types de données de base : int, float, string, boolean

devasc@labvm:~/labs/devnet-src$ **python3**

Python 3.8.2 (default, Mar 13 2020, 10:14:16)

[GCC 9.3.0] on linux

Tapez "help", "copyright", "credits" ou "license" pour plus d'informations.

>>> **type (98)**

<class 'int'>

>>> **type (98.6)**

<class 'float'>

>>> **type("Hi!")**

<class 'str'>

>>> **type(True)**

<class 'bool'>

### Passez en revue différents opérateurs booléens.

Le type de données booléen utilise les opérateurs indiqués dans le tableau.

| Opérateur | Signification |
| --- | --- |
| > | Supérieur à |
| < | Inférieur à |
| == | Égal à |
| != | Différent de |
| >= | Supérieur ou égal à |
| <= | Inférieur ou égal à |

Dans l'interpréteur, essayez les différents opérateurs booléens.

>>> **1<2**

Vrai

>>> **1<1**

Faux

>>> **1==1**

Vrai

>>> **1>=1**

Vrai

>>> **1<=1**

Vrai

### Utilisez l'interpréteur pour créer et utiliser une variable.

L'opérateur booléen permettant de déterminer si deux valeurs sont égales est le double signe égal (==). Un seul signe égal (=) est utilisé pour attribuer une valeur à une variable. La variable peut ensuite être utilisée dans d'autres commandes pour rappeler la valeur. Par exemple, créez et utilisez la variable suivante dans l'interpréteur interactif.

>>> **x=3**

>>> **x\*5**

15

>>> "Cisco"\*x

'CiscoCiscoCisco'

### Utilisez l'interpréteur pour concaténer plusieurs variables de chaîne.

La concaténation est le processus de combinaison de plusieurs chaînes en une seule chaîne. Par exemple, la concaténation de"**foot**" et "**ball**" est "**football**".

* + - 1. Entrez les quatre variables suivantes, puis concaténez-les ensemble dans une instruction **print ()** avec le signe plus (+). Notez que la variable d'espace a été définie pour être utilisée comme espace blanc entre les mots.

>>> **str1="Cisco"**

>>> **str2="Networking"**

>>> **str3="Academy"**

>>> **space=" "**

>>> **print(str1+space+str2+space+str3)**

Cisco Networking Academy

* + - 1. Pour imprimer les variables sans utiliser de variable pour créer l'espace, séparez les variables par une virgule.

>>> **impression (str1, str2, str3)**

Cisco Networking Academy

### Examen des différents types de données de coulée et d'impression.

* + - 1. La conversion entre les types de données est appelée type casting. Il est souvent nécessaire de procéder à des coulées de caractères afin de travailler avec différents types de données. Par exemple, la concaténation ne fonctionne pas lors de la jonction de différents types de données.

>>> **x=3**

>>> **print("The value of x is " + x)**

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: can only concatenate str (not "int") to str

>>>

* + - 1. Utilisez la fonction **str ()** pour convertir le type de données entier en type de données chaîne.

>>> **print("The value of x is " + str(x))**

The value of x is 3

>>> **type(x)**

<class 'int'>

* + - 1. Notez que le type de données de la variable x est toujours un entier. Pour convertir le type de données, réaffectez la variable au nouveau type de données.

>>> **x=str (x)**

>>> **type(x)**

<class 'str'>

* + - 1. Vous pouvez afficher un nombre flottant à un nombre spécifique de décimales au lieu du nombre complet. Pour ce faire, vous pouvez utiliser f-strings et la fonction **"{:.2f}".format**

**Remarque** : Cherchez sur Internet pour en savoir plus sur les f-strings et la fonction de format.

>>> **num = 22/7**

>>> **f"The value of num is {num}"**

'The value of num is 3.142857142857143'

>>> **pi = "{:.2f}".format(num)**

>>> **f"The value of pi is {pi}."**

'The value of pi is 3.14.'

>>>

## Listes de révision et dictionnaires

Dans cette partie, vous allez passer en revue les méthodes de création et de manipulation de listes et de dictionnaires.

### Créer et manipuler une liste.

* + - 1. En programmation, une variable de liste est utilisée pour stocker plusieurs éléments d'informations ordonnées. Les listes sont également appelées tableaux dans certains environnements de programmation.
* Créez une liste à l'aide de crochets **[ ]** et enfermant chaque élément de la liste avec des guillemets.
* Séparez les éléments par une virgule.
* Utilisez la commande **type ()** pour vérifier le type de données.
* Utilisez la commande **len ()** pour renvoyer le nombre d'éléments dans une liste.
* Appelez le nom de la variable de liste pour afficher son contenu.

L'exemple suivant montre comment créer une variable de liste appelée **hostnames**.

>>> **hostnames=["R1","R2","R3","S1","S2"]**

>>> **type(hostnames)**

<class 'list'>

>>> **len(hostnames)**

5

>>> **hostnames**

['R1', 'R2', 'R3', 'S1', 'S2']

* + - 1. Un élément d'une liste peut être référencé et manipulé à l'aide de son index.
* Le premier élément d'une liste est indexé comme zéro, le second est indexé comme un, et ainsi de suite.
* Le dernier élément peut être référencé avec l'index **[-1]**.
* Remplacez un élément en affectant une nouvelle valeur à l'index.
* Utilisez la commande **del** pour supprimer un élément d'une liste.

>>> **hostnames[0]**

'R1'

>>> **hostnames[-1]**

'S2'

>>> **hostnames[0]="RTR1"**

>>> **hostnames**

['RTR1', 'R2', 'R3', 'S1', 'S2']

>>> **del hostnames[3]**

>>> **hostnames**

['RTR1', 'R2', 'R3', 'S2']

>>>

### Créer et manipuler un dictionnaire.

* + - 1. Les dictionnaires sont des listes d'objets non ordonnées. Chaque objet contient une paire clé/valeur.
* Créez un dictionnaire à l'aide des accolades {}.
* Chaque entrée de dictionnaire comprend une clé et une valeur.
* Séparez une clé et sa valeur par un deux-points.
* Utilisez des guillemets pour les clés et les valeurs qui sont des chaînes.

Créez le dictionnaire suivant appelé **IPAddress** avec trois paires clé/valeur pour spécifier les valeurs d'adresse IP pour trois routeurs.

>>> **ipAddress={"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}**

>>> **type(ipAddress)**

<class 'dict'>

* + - 1. Contrairement aux listes, les objectifs à l'intérieur d'un dictionnaire ne peuvent pas être référencés par leur numéro de séquence. Au lieu de cela, vous référencez un objet dictionnaire en utilisant sa clé.
* La clé est entourée de crochets [ ].
* Les clés qui sont des chaînes peuvent être référencées à l'aide de guillemets simples ou doubles.
* Utilisez une **clé** dans l'instruction du **dictionnaire** pour vérifier si une clé existe dans le dictionnaire.
* Ajoutez une paire clé/valeur en définissant la nouvelle clé égale à une valeur.

>>> **ipAddress**

{'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1'}

>>> **ipAddress['R1']**

'10.1.1.1'

>>> **ipAddress["S1"]="10.1.1.10"**

>>> **ipAddress**

{'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1', 'S1': '10.1.1.10'}

>>>

* + - 1. Les valeurs d'une paire clé/valeur peuvent être n'importe quel autre type de données, y compris les listes et les dictionnaires. Par exemple, si R3 a plus d'une adresse IP, comment représentez-vous cela dans le dictionnaire **ipAddress**? Créez une liste pour la valeur de la clé R3.

>>> **ipAddress["R3"]=["10.3.3.1","10.3.3.2","10.3.3.3"]**

>>> **ipAddress**

{'S1': '10.1.1.10', 'R2': '10.2.2.1', 'R1': '10.1.1.1', 'R3': ['10.3.3.1', '10.3.3.2', '10.3.3.3']}

>>>

## Révision de la fonction d'entrée

Dans cette partie, vous allez examiner comment utiliser la fonction d'entrée pour stocker et afficher les données fournies par l'utilisateur.

### Créez une variable pour stocker l'entrée utilisateur, puis affichez la valeur.

La plupart des programmes nécessitent un type d'entrée à partir d'une base de données, d'un autre ordinateur, de clics de souris ou d'une entrée clavier. Pour l'entrée au clavier, utilisez la fonction **input ()** qui inclut un paramètre facultatif pour fournir une chaîne d'invite. Si la fonction d'entrée est appelée, le programme s'arrêtera jusqu'à ce que l'utilisateur fournisse l'entrée et frappe la touche Entrée. Attribuez la fonction **input ()** à une variable qui demande l'entrée à l'utilisateur, puis imprimez la valeur de l'entrée de l'utilisateur.

>>> firstName = input("What is your first name? ")

What is your first name? User\_Name

>>> print("Hello " + firstName +"!")

Hello User\_Name!

>>>

### Créez un script pour collecter des informations personnelles.

Créez et exécutez un script pour collecter des informations personnelles.

* + - 1. Ouvrez un fichier de script vide et enregistrez-le sous **personal-info.py** dans le dossier **~/labs/devnet-src/python**.
      2. Créez un script qui demande quatre éléments d'information tels que : prénom, nom, lieu et âge.
      3. Ajoutez une instruction d'impression qui combine toutes les informations en une seule phrase.
      4. Votre script doit s'exécuter sans aucune erreur, comme indiqué dans la sortie suivante.

devasc@labvm:~/labs/devnet-src$ **python3 person-info.py**

What is your first name? **Bob**

What is your last name? **Smith**

What is your location? **London**

What is your age? **36**

Hi Bob Smith! Your location is London and you are 36 years old.

devasc@labvm:~/labs/devnet-src$ ^C

## Examiner les fonctions if, for et while

Dans cette partie, vous examinez comment créer des instructions if ainsi que des boucles for et while.

### Créez une fonction if/else.

Dans la programmation, les instructions conditionnelles vérifient si quelque chose est vrai, puis exécutent des instructions basées sur l'évaluation. Si l'évaluation est fausse, différentes instructions sont effectuées.

* + - 1. Ouvrez un script vide et enregistrez-le sous **if-vlan.py**. Tapez le script suivant dans le fichier.

nativeVLAN = 1

DataVLAN = 100

if nativeVLAN == dataVLAN:

print("The native VLAN and the data VLAN are the same.")

else:

print("The native VLAN and the data VLAN are different.")

**Remarque** : En Python, utilisez quatre espaces pour mettre en retrait. Si vous enregistrez votre fichier dans VS Code, cette indention de quatre espaces sera automatique. Lorsque vous commencez l'instruction else, assurez-vous d'effectuer un retour arrière sur la marge gauche.

* + - 1. Enregistrez le script et exécutez-le. Votre sortie devrait ressembler à l'exemple suivant.

The native VLAN and the data VLAN are different.

* + - 1. Modifiez les variables afin que **NativeVLAN** et **DataVLAN** aient la même valeur. Enregistrez et exécutez à nouveau le script. Votre sortie devrait ressembler à l'exemple suivant.

The native VLAN and the data VLAN are the same.

### Créez une fonction if/elif/else.

Que se passe-t-il si nous avons plus de deux énoncés conditionnels à considérer ? Dans ce cas, nous pouvons utiliser des instructions **elif** au milieu de la fonction **if/else**. Une instruction **elif** est évaluée si l'instruction **if** est fausse et avant l'instruction **else**. Vous pouvez avoir autant d'instructions **elif** que vous le souhaitez. Cependant, la première correspondance sera exécutée et aucune des instructions **elif** restantes ne sera vérifiée. La déclaration **else** ne sera pas non plus.

Le script dans l'exemple suivant demande à l'utilisateur d'entrer le numéro d'une liste ACL IPv4, puis vérifie si ce nombre est une liste ACL IPv4 standard, une liste ACL IPv4 étendue ou une liste ACL IPv4 standard ou étendue.

* + - 1. Créez ce script pour vos fichiers. Ouvrez un script vide et enregistrez-le sous **if-acl.py**. Copiez le script dans le fichier.

aclNum = int(input("What is the IPv4 ACL number? "))

if aclNum >= 1 and aclNum <= 99:

print("This is a standard IPv4 ACL.")

elif aclNum >=100 and aclNum <= 199:

print("This is an extended IPv4 ACL.")

else:

print("This is not a standard or extended IPv4 ACL.")

**Remarque** : Le type de données de la fonction d'entrée est changé de la chaîne par défaut à un entier afin que les évaluations **if** et **elif** fonctionnent.

* + - 1. Exécutez plusieurs fois pour tester chaque instruction.

devasc@labvm:~/labs/devnet-src/python$ **python3 if-acl.py**

What is the IPv4 ACL number? **10**

This is a standard IPv4 ACL.

devasc@labvm:~/labs/devnet-src/python$ **python3 if-acl.py**

What is the IPv4 ACL number? **110**

This is an extended IPv4 ACL.

devasc@labvm:~/labs/devnet-src/python$ **python3 if-acl.py**

What is the IPv4 ACL number? **200**

Il ne s'agit pas d'une liste ACL IPv4 standard ou étendue.

devasc@labvm:~/labs/devnet-src/python$

### Créer une boucle de for.

La fonction Python **for** est utilisée pour boucler ou itérer à travers les éléments d'une liste ou effectuer une opération sur une série de valeurs.

* + - 1. Entrez ce qui suit dans l'interpréteur interactif pour voir comment fonctionne une boucle for. Le nom de la variable **item** est arbitraire et peut être tout ce que le programmeur choisit. Souvent, les programmeurs raccourcissent cela à la lettre **i**.

**Remarque** : Assurez-vous d'entrer quatre espaces pour mettre en retrait la fonction **print ()** et appuyez deux fois sur la touche Entrée pour quitter la boucle **for**.

devasc@labvm:~/labs/devnet-src/python$ **python3**

Python 3.8.2 (default, Mar 13 2020, 10:14:16)

[GCC 9.3.0] on linux

Tapez "help", "copyright", "credits" ou "license" pour plus d'informations.>>> **devices=["R1","R2","R3","S1","S2"]**

>>> **for item in devices:**

... **print(item)**

…

R1

R2

R3

S1

S2

>>>

* + - 1. Que se passe-t-il si vous ne voulez lister que les éléments qui commencent par la lettre R ? Une instruction **if** peut être incorporée dans une boucle **for** pour y parvenir. En continuant avec la même instance Python, entrez ce qui suit dans l'interpréteur interactif.

**Remarque** : Assurez-vous d'entrer quatre espaces pour mettre en retrait la fonction **if** et les huit espaces pour mettre en retrait la fonction **print ()**. Appuyez deux fois sur la touche Entrée pour quitter et exécuter la boucle for.

>>> **for item in devices:**

... **if "R" in item:**

... **print(item)**

…

R1

R2

R3

>>>

* + - 1. Vous pouvez également utiliser une combinaison de la boucle **for** et de la déclaration **if** pour créer une nouvelle liste. Entrez l'exemple suivant pour voir comment utiliser la méthode **append ()** pour créer une liste appelée switches. Veillez à respecter les exigences d'indention.

>>> **switches=[]**

>>> **for item in devices:**

... **if "S" in item:**

... **switches.append(item)**

…

>>> **switches**

['S1', 'S2']

>>>

### Boucle while

Au lieu d'exécuter un bloc de code une fois, comme dans une instruction **if**, vous pouvez utiliser une boucle **while**. Une boucle while continue d'exécuter un bloc de code tant qu'une expression booléenne reste vraie. Cela peut entraîner l'exécution infini d'un programme si vous ne vous assurez pas que votre script inclut une condition pour que la boucle while s'arrête. Alors que les boucles ne s'arrêteront pas tant que l'expression booléenne n'est pas évaluée comme false.

* + - 1. Ouvrez un script vide et enregistrez-le sous **while-loop.py**. Utilisez une boucle while pour créer le programme suivant qui compte d'une valeur programmée jusqu'à un nombre fourni par l'utilisateur. Le programme effectue les opérations suivantes :
* Demande à l'utilisateur un numéro.
* Casse la valeur de chaîne entrée à un entier : **x = int (x)**.
* Définit une variable pour démarrer le compte : **y = 1**.
* Alors que **y <= x**, imprime la valeur de y et incrémente y de 1.

x=input("Enter a number to count to: ")

x=int (x)

y=1

x=input("Enter a number to count to: ")

print(y)

y=y+1

* + - 1. Enregistrez et exécutez votre script. Vous devriez obtenir la sortie comme suit.

devasc@labvm:~/labs/devnet-src/python$ **python3 while-loop.py**

Entrez un nombre à compter : **10**

1

2

3

4

5

6

7

8

9

10

devasc@labvm:~/labs/devnet-src/python$

* + - 1. Au lieu d'utiliser while **y <= x**, nous pouvons modifier la boucle while pour utiliser une vérification booléenne et casser pour arrêter la boucle lorsque la vérification est évaluée comme false. Modifiez le script **while-loop.py** comme indiqué dans ce qui suit :

x=input("Enter a number to count to: ")

x=int (x)

y=1

while True:

print(y)

y=y+1

if y>x:

break

* + - 1. Enregistrez et exécutez votre script. Vous devriez obtenir la même sortie que dans l'étape 4b ci-dessus.

### Utilisez une boucle while pour vérifier la présence d'une commande de fermeture utilisateur.

Que faire si nous voulons que le programme s'exécute autant de fois que l'utilisateur le souhaite jusqu'à ce que l'utilisateur quitte le programme ? Pour ce faire, nous pouvons intégrer le programme dans une boucle while qui vérifie si l'utilisateur entre une commande quit, telle que **q** ou **quit**.

* + - 1. Modifiez votre script **while-loop.py** avec les modifications suivantes :
* Ajoutez une autre boucle while au début du script qui vérifiera une commande quit.
* Ajoutez une fonction if à la boucle while pour vérifier **q** ou **quitter**.

**Remarque** : Assurez-vous d'inclure tous les niveaux d'indention requis.

while True:

x=input("Enter a number to count to: ")

if x == 'q' or x == 'quit':

break

x=int (x)

y=1

while True:

print(y)

y=y+1

if y>x:

break

* + - 1. Enregistrez et exécutez votre script. Votre sortie devrait ressembler à la suivante dans laquelle l'utilisateur a entré deux valeurs différentes avant de quitter le programme.

devasc@labvm:~/labs/devnet-src/python$ **python3 while-loop.py**

Entrez un nombre à compter : **3**

1

2

3

Enter a number to count to: **5**

1

2

3

4

5

Enter a number to count to: **quit**

devasc@labvm:~/labs/devnet-src/python$

## Méthodes d'examen pour l'accès aux fichiers

Dans cette partie, vous passez en revue les méthodes d'accès, de lecture et de manipulation d'un fichier.

### Créez un programme qui lit un fichier externe.

En plus de la saisie utilisateur, vous pouvez accéder à une base de données, à un autre programme informatique ou à un fichier pour fournir une entrée à votre programme. La fonction **open ()** peut être utilisée pour accéder à un fichier en utilisant la syntaxe suivante :

**open**(*name*, [*mode*])

Le paramètre name est le nom du fichier à ouvrir. Si le fichier se trouve dans un répertoire différent de celui de votre script, vous devrez également fournir des informations sur le chemin d'accès. Pour nos besoins, nous ne sommes intéressés que par trois paramètres de mode :

* **r** - lire le fichier (mode par défaut si le mode est omis).
* **w** - écrire sur le fichier, en remplaçant le contenu du fichier.
* **a** - ajouter au fichier.
  + - 1. Ouvrez un script vide et enregistrez-le sous **file-access.py**.
      2. Créez le programme suivant pour lire et imprimer le contenu du fichier **devices.txt** qui se trouve dans le même répertoire que votre programme. Après avoir imprimé le contenu du fichier, utilisez la fonction **close ()** pour le supprimer de la mémoire de l'ordinateur.

file=open("devices.txt","r")

for item in file:

print(item)

file.close()

**Remarque** : Le contenu du fichier est défini sur une variable nommée fichier. Cependant, cette variable peut être appelée tout ce que le programmeur choisit.

* + - 1. Enregistrez et exécutez votre programme. You should get output similar to the following.

devasc@labvm:~/labs/devnet-src/python$ **python3 file-access.py**

Cisco 819 Router

Cisco 881 Router

Cisco 888 Router

Cisco 1100 Router

Cisco 4321 Router

Cisco 4331 Router

Cisco 4351 Router

Cisco 2960 Catalyst Switch

Cisco 3850 Catalyst Switch

Cisco 7700 Nexus Switch

Cisco Meraki MS220-8 Cloud Managed Switch

Cisco Meraki MX64W Security Appliance

Cisco Meraki MX84 Security Appliance

Cisco Meraki MC74 VoIP Phone

Cisco 3860 Catalyst Switch

devasc@labvm:~/labs/devnet-src/python$

### Supprimez les lignes vides de la sortie.

Vous avez peut-être remarqué que Python a ajouté une ligne vide après chaque entrée. Nous pouvons supprimer cette ligne vide en utilisant la méthode **strip ()**.

* + - 1. Modifiez votre programme **file-access.py** pour inclure la méthode **strip ()**.

file=open("devices.txt","r")

for item in file:

item=item.strip ()

print(item)

file.close()

* + - 1. Enregistrez et exécutez votre programme. Vous devriez obtenir un résultat similaire à ce qui suit.

devasc@labvm:~/labs/devnet-src/python$ **python3 file-access.py**

Cisco 819 Router

Cisco 881 Router

Cisco 888 Router

Cisco 1100 Router

Cisco 4321 Router

Cisco 4331 Router

Cisco 4351 Router

Cisco 2960 Catalyst Switch

Cisco 3850 Catalyst Switch

Cisco 7700 Nexus Switch

Cisco Meraki MS220-8 Cloud Managed Switch

Cisco Meraki MX64W Security Appliance

Cisco Meraki MX84 Security Appliance

Cisco Meraki MC74 VoIP Phone

Cisco 3860 Catalyst Switch

devasc@labvm:~/labs/devnet-src/python$

### Copiez le contenu d'un fichier dans une variable de liste.

La plupart du temps, lorsque les programmeurs accèdent à une ressource externe telle qu'une base de données ou un fichier, ils veulent copier ce contenu dans une variable locale qui peut ensuite être référencée et manipulée sans affecter la ressource d'origine.

Le fichier **devices.txt** est une liste de périphériques Cisco qui peuvent facilement être copiés dans une liste Python à l'aide des étapes suivantes :

* Créer une liste vide.
* Utilisez le paramètre **append** pour copier le contenu du fichier dans la nouvelle liste.
* Imprimer la liste.
  + - 1. Modifiez votre file-access.py comme indiqué dans le suivant

devices=[]

file=open("devices.txt","r")

for item in file:

item=item.strip ()

devices.append (élément)

file.close()

print(devices)

* + - 1. Enregistrez et exécutez votre programme. Vous devriez obtenir un résultat similaire à ce qui suit.

devasc@labvm:~/labs/devnet-src/python$ **python3 file-access.py**

['Cisco 819 Router', 'Cisco 881 Router', 'Cisco 888 Router', 'Cisco 1100 Router', 'Cisco 4321 Router', 'Cisco 4331 Router', 'Cisco 4351 Router', 'Cisco 2960 Catalyst Switch', 'Cisco 3850 Catalyst Switch', 'Cisco 7700 Nexus Switch', 'Cisco Meraki MS220-8 Cloud Managed Switch', 'Cisco Meraki MX64W Security Appliance', 'Cisco Meraki MX84 Security Appliance', 'Cisco Meraki MC74 VoIP Phone', 'Cisco 3860 Catalyst Switch']

devasc@labvm:~/labs/devnet-src/python$

### Défi : Créez un script pour permettre à l'utilisateur d'ajouter des périphériques.

Que faire si vous voulez ajouter d'autres périphériques au fichier **devices.txt**  ? Vous pouvez créer un programme pour ouvrir le fichier en mode Ajout, puis demander à l'utilisateur de fournir le nom des nouveaux périphériques. Suivez les étapes suivantes pour créer un scénario :

* + - * 1. Ouvrez un nouveau fichier et enregistrez-le sous **file-access-input.py**.
        2. Pour la fonction **open ()** utilisez le mode **a**, qui vous permettra d'ajouter un élément au fichier **devices.txt**.
        3. Dans une boucle **while True :**, incorporez une commande de fonction **input ()** qui demande à l'utilisateur le nouveau périphérique.
        4. Définissez la valeur de l'entrée de l'utilisateur sur une variable nommée **NewiTem**.
        5. Utilisez une instruction **if** qui casse la boucle si l'utilisateur tape **exit** et imprime l'instruction "All done!".
        6. Utilisez la commande **file.write(newItem + “\n”)** pour ajouter le nouveau périphérique fourni par l'utilisateur.
        7. Fermez le fichier pour le libérer de la mémoire de l'ordinateur.

Exécutez et dépannez votre script jusqu'à ce que vous obteniez une sortie similaire à la suivante.

devasc@labvm:~/labs/devnet-src/python$ **python3 file-access-input.py**

Enter device name: **Cisco 1941 Router**

Enter device name: **Cisco 2950 Catalyst Switch**

Enter device name: **exit**

All done!

devasc@labvm:~/labs/devnet-src/python$

Fin du document