Laboratorio - Revisión de programación de Python.

Objetivos

Parte 1: Ejecutar la máquina virtual (Virtual Machine) de DEVASC.

Parte 2: Iniciar Python y código VS.

Parte 3: Revisar tipos de datos y variables.

Parte 4: Listas de revisión y diccionarios.

Parte 5: Revisar la función de entrada.

Parte 6: Revisar las funciones if, for y while.

Parte 7: Métodos de revisión para el acceso a archivos

Aspectos básicos/Situación.

En este laboratorio, se revisan las habilidades básicas de programación de Python, incluyendo tipos de datos, variables, listas, diccionarios, entrada de usuario, sentencias if, bucles for y while y acceso a archivos. Este laboratorio no es un sustituto de la experiencia de programación previa y no cubre necesariamente todas las habilidades de Python que necesitará para este curso. Sin embargo, este laboratorio debería servir como una buena medida de sus habilidades de programación de Python y ayudarlo a dirigirlo a donde puede necesitar más revisión.

Nota: Este es un recordatorio. Asegúrese de observar bien los ajustes correctos de la sangría en Python al escribir sus scripts. Si necesita un tutorial, busque en Internet Reglas de sangría de Python.

Recursos necesarios.

- Una computadora con el sistema operativo de su elección.
- VirtualBox o VMware.
- Máquina virtual (Virtual Machine) DEVASC.

Instrucciones

Parte 1: Ejecute la máquina virtual (virtual machine) de DEVASC.

Si aún no ha completado el **Laboratorio: Instale el Entorno de Laboratorio de Máquina Virtual**, hágalo hora. Si ya ha completado ese laboratorio, ejecute la **máquina virtual de DEVASC**.

Parte 2: Comenzar con el código Python y VS.

En esta parte, se va revisar el inicio del intérprete interactivo de Python y el uso de Visual Studio Code para escribir y ejecutar un script "Hola Mundo"

Paso 1: Iniciar Python.

a. Para verificar la versión de Python que se ejecuta en la máquina virtual (virtual machine), abra una ventana de terminal e ingrese el comando **python3 -V**. Este es un buen comando para ejecutar si va a instalar Python en un equipo diferente o necesitara verificar qué versión está instalada.

```
devasc @labvm: ~$ python3 -V
Python 3.8.2
```

Nota: Tiene que cambiarlo a **python2 -V** en caso de que esté usando un dispositivo diferente y éste esté ejecutando la versión 2. Sin embargo, a partir del 1 de enero de 2020, Python 2 ya no va a recibir soporte. Por lo tanto, Python 2 no es compatible con este laboratorio o este curso.

Nota: En el momento en que se escribió este laboratorio, Python 3.8.2 era la última versión. Aunque puede actualizar su instalación de Python con el comando **sudo apt-get install python3**, ya que este laboratorio y el resto de los laboratorios de este curso se basan en Python 3.8.2.

b. Para iniciar Python, escriba **python3**. Los tres corchetes angulares (>>>) indican que está en el intérprete interactivo de Python.

```
devasc @labvm ~$ python3
Python 3.8.2 (predeterminado, 13 de marzo de 2020, 10:14:16)
[GCC 9.3.0] en Linux
Escriba "help", "copyright", "credit" o "license" para obtener más información.
>>>
```

Paso 2: Use el Intérprete como calculadora.

a. Desde aquí, puede realizar una variedad de tareas básicas de programación, incluyendo operaciones matemáticas. La tabla muestra la sintaxis de Python que se utilizará para las operaciones matemáticas más comunes.

Operación.	Matemáticas.	Sintaxis.
Suma.	a+b	a+b
Resta.	a-b	a-b
Multiplicación.	axb	a*b
División.	a÷b	a/b
Exponentes.	a^b	a**b

Introduzca algunas operaciones matemáticas utilizando la sintaxis de Python, como se muestra en los ejemplos.

```
>>> 2+3
5
>>> 10-4
6
>>> 2*4
8
>>> 20/5
4,0
>>> 3**2
9
```

b. Recuerde que Python utiliza el orden estándar de operaciones comúnmente conocido como PEMDAS. Las expresiones matemáticas se evalúan en el siguiente orden.

Parentesis

Exponentes

Multiplicación y División

Suma y Resta

Intente introducir una expresión con un orden complejo de operaciones en el intérprete interactivo.

Paso 3: Use el intérprete interactivo para hacer print a una string.

Una string es cualquier secuencia de caracteres como letras, números, símbolos o signos de puntuación. El intérprete interactivo mostrará directamente el texto que introduzca como una string, siempre y cuando la escriba entre comillas simples (') o comillas dobles (").

a. Escriba "¡Hola Mundo!" o 'Hola Mundo! ' en el intérprete interactivo.

```
>>> "Hola Mundo! "
'Hola Mundo! !'
>>> 'Hola Mundo! '
'Hola Mundo! !'
```

b. El comando **print** también se puede utilizar directamente en el intérprete interactivo.

```
>>> print("Hola Mundo! ")
Hola Mundo!
```

c. Para salir del intérprete interactivo, escriba **quit ()**.

```
>>> quit()
devasc @labvm: ~$
```

Paso 4: Abra Código VS y cree un script para Hello World.

Hay muchos entornos de desarrollo disponibles para que los programadores administren sus proyectos de codificación. En este curso, los laboratorios usarán la instalación de la máquina virtual (virtual machine), empleando Visual Studio Code de Microsoft (Código VS).

a. Abra el código VS. Si esta es su primera vez ingresando, lo más probable es que se le presente una ventana de **bienvenida**.

b. No dude en explorar los menús y opciones de Código VS en su tiempo libre. Por ahora, haga clic en **Archivo > Nuevo archivo** para abrir un nuevo archivo.

c. En el nuevo archivo, escriba el comando **print** del paso anterior.

d. Guarde el script como **hello-world.py** en la carpeta **labs/devnet-src/python**. Asegúrese de agregar la extensión **.py** para el archivo Python.

e. Para ejecutar el script, haga clic en **Ejecutar > Ejecutar sin depuración**. Una ventana de terminal se abre dentro del Código VS, ejecute el código para iniciar una instancia de Python, ejecute su script y luego sale de Python de nuevo a la línea de comandos de 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$
```

f. Ahora que tiene una línea de comando abierta dentro de VS Code, puede iniciar Python manualmente y ejecutar rápidamente su script con el siguiente comando.

```
devasc @labvm: ~/labs/devnet-src/python$ python3 hello-world.py
Hello World!
devasc @labvm: ~/labs/devnet-src/python$
```

g. También puede abrir una ventana de terminal fuera de VS Code e introducir el mismo comando asegurándose de proporcionar información de ruta.

```
devasc @labvm: ~$ python3 ~/labs/devnet-src/python/hello-world.py
Hello World!
```

```
devasc @labvm: ~$
```

En este curso, el usuario, normalmente, ejecutará sus scripts directamente dentro del Código VS.

Parte 3: Revisar tipos de datos y variables

En esta parte, se va a hacer uso del intérprete interactivo para revisar tipos de datos, crear variables, concatenar strings y emitir entre algunos tipos de datos.

Paso 1: Utilice el intérprete interactivo para revisar los tipos de datos básicos.

En programación, los tipos de datos son una clasificación que le dice al intérprete cómo el programador pretende utilizar los datos. Por ejemplo, el intérprete necesita saber si los datos introducidos por el programador son un número o una string (cadena). Aunque hay varios tipos de datos diferentes, nos centraremos solo en los siguientes:

- **Integer(Entero)** : Se utiliza para especificar números enteros (sin decimales), como 1, 2, 3, etc. Si se introduce un entero con un decimal, el intérprete ignora el decimal. Por ejemplo, 3.75 se interpreta como 3.
- **Float (Flotante)** : Se utiliza para especificar números que necesitan un valor decimal, como 3.14159.
- **String (Cadena)** : Cualquier secuencia de caracteres como letras, números, símbolos o signos de puntuación.
- **Boolean (Booleano)** : Cualquier tipo de datos que tenga un valor de Verdadero o Falso.

Utilice el **tipo()**de comando para determinar los tipos básicos de datos: int, float, string, boolean.

```
devasc @labvm: ~/labs/devnet-src$ python3
Python 3.8.2 (predeterminado, 13 de marzo de 2020, 10:14:16)
[GCC 9.3.0] en Linux
Escriba "help", "copyright", "credit" o "license" para obtener más información.
>>> type(98)
<class 'int'>
>>> type(98.6)
<class 'float'>
>>> type("Hi!")
<class 'str'>
>>> type(True)
<class 'bool'>
```

Paso 2: Revisar diferentes operadores booleanos.

El tipo de datos booleano utiliza los operadores que se muestran en la tabla.

Operador.	Significado.
>	Mayor que
<	Menor que
==	Igual a
!=	No equivalente a
> =	Mayor que o equivalente a
< =	Menor que o equivalente a

En el intérprete, pruebe los diferentes operadores booleanos.

```
>>> 1<2
Verdadero
>>> 1<1
Falso
>>> 1==1
Verdadero
>>> 1>=1
Verdadero
>>> 1<=1
Verdadero
```

Paso 3: Utilice el intérprete para crear y utilizar una variable.

El operador booleano determina si dos valores son iguales, con el signo doble igual (==). Un único signo igual (=) se utiliza para asignar un valor a una variable. La variable se puede utilizar en otros comandos para hacer volver un valor. Por ejemplo, cree y utilice la siguiente variable en el intérprete interactivo.

```
>>> x=3
>>> x*5
15
>>> "Cisco"*X
'CiscoCiscoCisco'
```

Paso 4: Utilice el intérprete para concatenar múltiples variables tipo string.

La concatenación es el proceso de combinar múltiples strings (cadenas) en una sola string. Por ejemplo, la concatenación de "**pie**" y "**bola**" es "**fútbol**"

a. Introduzca las siguientes cuatro variables y, a continuación, concaténelas juntas en una instrucción **print ()** con el signo más (+). Observe que la variable de espacio se definió para su uso como espacio en blanco entre las palabras.

```
>>> str1="Cisco"
>>> str2="Networking"
>>> str3="Academy"
>>> space=" "
>>> print(str1+space+str2+space+str3)
Cisco Networking Academy
```

b. Para imprimir variables sin necesidad de usar una variable para crear un espacio, debe de separar las variables con una coma.

```
>>> print (str1, str2, str3)
Cisco Networking Academy
```

Paso 5: Revisión de emisión e impresión de diferentes tipos de datos.

a. La conversión entre tipos de datos se denomina conversión de tipo. La conversión de tipos a menudo se necesita para poder trabajar con diferentes tipos de datos. Por ejemplo, la concatenación no funciona cuando se unen diferentes tipos de datos.

```
>>> x=3
>>> print("El valor de x es" + x)
Traceback (última llamada más reciente):
```

```
  File "<stdin>", line 1, in <module>
TypeError: Solo puede concatenar str (no "int") a str
>>>
```

b. Utilice la función **str ()** para convertir el tipo de datos integer (entero) en un tipo de datos string (cadena).

```
>>> print("El valor de x es " + str(x))
El valor de x es 3.
>>> type(x)
<class 'int'>
```

c. Observe que el tipo de datos de la variable x sigue siendo un entero. Para convertir el tipo de datos, reasigne la variable al nuevo tipo de datos.

```
>>> x=str (x)
>>> type(x)
<class 'str'>
```

d. Probablemente, deseé mostrar una variable float (flotante) con un número específico de decimales, en lugar de un número entero. Para hacer esto, puede usar f-strings y la función **"{:.2f}" .format**.

Nota: Busque en Internet para obtener más información sobre las f-strings y la función de formato.

```
>>> num = 22/7
>>> f"El valor de num es {num}"
'El valor de num es 3.142857142857143'
>>> pi = "{:.2f}" .format (num)
>>> f"El valor de pi es  {pi}."
'El valor de pi es 3,14. '
>>>
```

Parte 4: Listas de revisión y diccionarios

En esta parte, revisará los métodos para crear y manipular listas y diccionarios.

Paso 1: Crear y manipular una lista.

a. En programación, una variable de lista se utiliza para almacenar múltiples piezas de información ordenada. Las listas también se denominan matrices (arrays) en algunos entornos de programación.

- Cree una lista usando corchetes **[]** y adjuntando cada elemento de la lista con comillas.
- Separe los ítems con una coma.
- Utilice el comando **type ()** para verificar el tipo de datos.
- Utilice el comando **len ()** para devolver el número de elementos de una lista.
- Emita el nombre de la variable de lista para mostrar su contenido.

En el siguiente ejemplo se muestra cómo crear una variable de lista denominada **hostnames**.

```
>>> hostnames= ["R1", "R2", "R3", "S1", "S2"]
>>> type(hostnames)
<class 'list'>
>>> len(hostnames)
5
>>> hostnames
['R1', 'R2', 'R3', 'S1', 'S2']
```

b. Un elemento de una lista puede ser referenciado y manipulado utilizando su índice.

- El primer elemento de una lista se indexa como cero, el segundo se indexa como uno, y así sucesivamente.
- Se puede hacer referencia al último elemento con el índice **[-1]**.
- Reemplace un elemento asignando un nuevo valor al índice.
- Utilice el comando **del** para eliminar un elemento de una lista.

```
>>> hostnames[0]
'R1'
>>>hostnames[-1]
'S2'
>>>hostnames[0]="RTR1"
>>> hostnames
['RTR1', 'R2', 'R3', 'S1', 'S2']
>>> del hostnames[3]
>>> hostnames
['RTR1', 'R2', 'R3', 'S2']
>>>
```

Paso 2: Crear y manipular un diccionario.

a. Los diccionarios son listas desordenadas de objetos. Cada objeto contiene un par clave/valor.

- Cree un diccionario usando las llaves {}.
- Cada entrada del diccionario incluye una clave y un valor.
- Separe una clave y su valor con dos puntos.
- Utilice comillas para claves y valores que son strings (cadenas).

Cree el siguiente diccionario denominado **ipAddress** con tres pares clave/valor para especificar los valores de dirección IP para tres routers.

```
>>> ipAddress= {"R1":" 10.1.1.1", "R2": "10.2.2.1", "R3":" 10.3.1"}
>>> type(ipAddress)
<class 'dict'>
```

b. A diferencia de las listas, los objetivos dentro de un diccionario no pueden ser referenciados por su número de secuencia. En su lugar, se hace referencia a un objeto de diccionario mediante su clave.

- La clave está entre corchetes [].
- Las claves que son cadenas pueden ser referenciadas usando comillas simples o dobles.
- Utilice una **clave** en la instrucción del **diccionario** para verificar si existe alguna clave en el diccionario.
- Agregue un par clave/valor estableciendo la nueva clave igual a un valor.

```
>>> 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'}
```

```
>>>
```

c. Los valores de un par clave/valor pueden ser cualquier otro tipo de datos, incluidas listas y diccionarios. Por ejemplo, si R3 tiene más de una dirección IP, ¿cómo representaría eso dentro del diccionario **IP Address** ? Cree una lista para el valor de la clave 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']}
>>>
```

Parte 5: Revisar la función de entrada.

En esta parte, revisará cómo utilizar la función de entrada para almacenar y mostrar los datos proporcionados por el usuario.

Paso 1: Crear una variable para almacenar la entrada del usuario y, a continuación, muestre el valor.

La mayoría de los programas requieren algún tipo de entrada, ya sea de una base de datos, de otra computadora, de clics del mouse o de entrada del teclado. Para la entrada del teclado, utilice la función **input ()** que incluye un parámetro opcional para proporcionar una cadena de mensaje (prompt string). Si se llama a la función de entrada, el programa se detendrá hasta que el usuario proporcione la entrada y presione la tecla Enter. Asigne la función **input ()** a una variable que solicite al usuario la entrada y luego imprima el valor de la entrada del usuario.

```
>>> firstName = input("¿Cuál es tu primer nombre? ")
¿Cuál es tu primer nombre? User_Name
>>> print("Hola " + firstName +"!")
Hola User_Name!
>>>
```

Paso 2: Cree un script para recopilar información personal.

Cree y ejecute un script para recopilar información personal.

a. Abra un archivo de script en blanco y guárdelo como **personal-info.py** en la carpeta **~/labs/devnet-src/python**.

b. Cree un script que solicite cuatro elementos de información como: nombre, apellido, ubicación y edad.

c. Agregue una instrucción de print (impresión) que combine toda la información en una frase.

d. Su script debe ejecutarse sin ningún error, como se muestra en la siguiente salida.

```
devasc @labvm: ~/labs/devnet-src$ python3 person-info.py
¿Cuál es tu primer nombre? Bob
¿Cuál es tu apellido? Smith
¿Cuál es tu localización? Londres
¿Cuál es su edad? 36
¡Hola Bob Smith! Tu ubicación es Londres y tienes 36 años.
devasc @labvm: ~/labs/devnet-src$ ^C
```

personal-info.py

```
firstName = input("Cual es tu nombre? ")
lastName = input("Cual es tu apellido? ")
location = input("Cual es tu localización? ")
```

```
age = input("Cual es tu edad? ")
print("Hola " + firstName, lastName + "! Tu ubicación es "+ location +" y tienes "+ age +" años.")
```

Parte 6: Revisar las funciones If, For y While

En esta parte, revisa cómo crear declaraciones if, así como bucles for y while.

Paso 1: Cree una función if/else.

En programación, las declaraciones condicionales verifican si algo es cierto y luego llevan a cabo instrucciones basadas en la evaluación. Si la evaluación es falsa, se llevan a cabo diferentes instrucciones.

a. Abra un script en blanco y guárdelo como **if-vlan.py**. Escriba la siguiente secuencia de comandos en el archivo.

```
nativeVLAN = 1
DataVLAN = 100
if nativeVLAN == DataVLAN:
    print("La VLAN nativa y los datos VLAN son iguales.")
else:
    print("La VLAN nativa y los datos VLAN son diferentes.")
```

Nota: En Python, use cuatro espacios para la sangría. Si guarda su archivo en Código VS, la sangría de cuatro espacios será automática. Al comenzar la instrucción else, asegúrese de retroceder al margen izquierdo.

b. Guarde el script y ejecútelo. Su salida debería ser similar al siguiente ejemplo.

```
La VLAN nativa y la VLAN de datos son diferentes.
```

c. Modifique las variables para que **NativeVLAN** y **DataVLAN** para que tengan el mismo valor. Guarde y ejecute el script de nuevo. Su salida debería ser similar al siguiente ejemplo.

```
La VLAN nativa y la VLAN de datos son las mismas.
```

Paso 2: Cree una función if/elif/else.

¿Qué pasa si tenemos más de dos declaraciones condicionales que considerar? En este caso, podemos usar declaraciones **elif** en el medio de la función **if/else**. Una sentencia **elif** se evalúa si la sentencia **if** es falsa y si se encuentra antes de la sentencia **else**. Puede tener tantas declaraciones **elif** como desee. Sin embargo, se ejecutará la primera coincidencia y no se comprobará ninguna de las sentencias **elif** restantes. Tampoco lo hará la declaración **else**.

El script del siguiente ejemplo, pide al usuario que introduzca el número de una ACL IPv4 y, a continuación, comprueba si ese número es una ACL IPv4 estándar, ACL IPv4 extendida o ninguna ACL IPv4 estándar o extendida.

a. Cree este script para sus archivos. Abra un script en blanco y guárdelo como **if-acl.py**. Copie el script en el archivo.

```
aclNum = int(input("Cual es el número IPV4 ACL? "))
if aclNum >= 1 and aclNum <= 99:
    print("Este es un ACL IPv4 estándar.")
elif aclNum >=100 and aclNum <= 199:
    print("Este es una ACL IPv4 extendida")
else:
    print("Esta ACL IPv4 no es extendida o estandar .")
```

Nota: El tipo de datos para la función de entrada se cambia de la string predeterminada a un integer (entero) para que las evaluaciones if y elif funcionen.

b. Ejecute varias veces para probar cada instrucción.

```
devasc @labvm: ~/labs/devnet-src/python$ python3 if-acl.py
¿Qué es el número ACL IPv4? 10
Esta es una ACL IPv4 estándar.
devasc @labvm: ~/labs/devnet-src/python$ python3 if-acl.py
¿Qué es el número ACL IPv4? 110
Esta es una ACL IPv4 extendida.
devasc @labvm: ~/labs/devnet-src/python$ python3 if-acl.py
¿Qué es el número ACL IPv4? 200
Esta no es una ACL IPv4 estándar o extendida.
devasc @labvm: ~/labs/devnet-src/python$
```

Paso 3: Cree un bucle for.

La función de Python **for** se utiliza para crear un bucle o iterar a través de los elementos de una lista o realizar una operación en una serie de valores.

a. Introduzca lo siguiente en el intérprete interactivo para ver cómo funciona un bucle for. El nombre de la variable **Elemento** es arbitrario, ya que puede ser cualquier cosa que el programador elija. A menudo, los programadores acortarán esto a la letra **i**.

Nota: Asegúrese de introducir cuatro espacios para la sangría de la función print () y pulse la tecla Enter dos veces para salir del bucle for.

```
devasc @labvm: ~/labs/devnet-src/python$ python3
Python 3.8.2 (predeterminado, 13 de marzo de 2020, 10:14:16)
[GCC 9.3.0] en Linux
Escriba "help", "copyright", "credits" o "license" para obtener más información.>>>
devices= ["R1", "R2", "R3", "S1", "S2"]
>>> for item in devices:
... print(item)
...
R1
R2
R3
S1
S2
>>>
```

b. ¿Qué pasa si el usuario solo desea en-listar los elementos que comiencen con la letra R? Una sentencia **if** se puede incrustar en un bucle **for** para lograr eso. Continuando con la misma instancia de Python, escriba lo siguiente en el intérprete interactivo.

Nota: Asegúrese de introducir cuatro espacios para la sangría de función if y los ocho espacios para la sangría de la **función print (). Pulse la tecla Enter dos veces para salir y ejecutar el bucle for.**

```
>>> for item in devices:
... if "R" in item:
... print(item)
...
R1
```

```
R2
R3
>>>
```

c. También puede utilizar una combinación del bucle **for** y la sentencia **if** para crear una nueva lista. Introduzca el siguiente ejemplo para ver cómo utilizar el método **append ()** para crear una nueva lista denominada switches. Asegúrese de seguir los requisitos para la sangría.

```
>>> switches=[]
>>> for item in devices:
... if "S" in item:
... switches.append (item)
...
>>> switches
['S1', 'S2']
>>>
```

Paso 4: Bucle While.

En lugar de ejecutar un bloque de código una vez, como en una sentencia **if**, puede usar un bucle **while**. Un bucle while sigue ejecutando un bloque de código siempre y cuando una expresión booleana siga siendo verdadera. Esto puede hacer que un programa se ejecute sin fin si no se asegura de que el script incluya una condición para que el bucle while se detenga. Los bucles While no se detendrán hasta que la expresión booleana se evalúe como falso.

a. Abra un script en blanco y guárdelo como **while-loop.py**. Utilice un bucle while para crear el siguiente programa que cuenta desde un valor programado hasta un número proporcionado por el usuario. El programa hace lo siguiente:

- Pide al usuario un número.
- Emite el valor de la string de entrada a un integer: **x = int (x)**.
- Establece una variable para iniciar el recuento **: y = 1**.
- El bucle While **y <= x**, imprime el valor de y e incrementa y por 1.

```
x=input("Ingrese un número para contar hasta: ")
x=int (x)
y=1
while y<=x:
    print(y)
    y=y+1
```

b. Guarde y ejecute el script. Su salida debe de quedar como el siguiente ejemplo:

```
devasc @labvm: ~/labs/devnet-src/python$ python3 while-loop.py
Introduzca un número para contar: 10
1
2
3
4
5
6
7
8
9
```

```
10
devasc @labvm: ~/labs/devnet-src/python$
```

c. En lugar de usar while **y <= x**, podemos modificar el bucle while para usar una comprobación booleana y romper para detener el bucle cuando la comprobación se evalúa como falsa. Modifique el script **while-loop.py** como se muestra en lo siguiente:

```
x=input("Ingrese un número para contar hasta:")
x=int (x)
y=1
while True:
    print(y)
    y=y+1
    si y>x:
        break
```

d. Guarde y ejecute el script. Debería obtener el mismo resultado que en el paso 4b anterior.

Paso 5: Utilice un bucle while para comprobar si hay un comando quit de usuario.

¿Qué pasa si queremos que el programa se ejecute tantas veces como el usuario desee hasta que el usuario salga del programa? Para ello, podemos incrustar el programa en un bucle while que comprueba si el usuario ingresa un comando quit, como **q** o **quit**.

a. Modifique su script **while-loop.py** con los siguientes cambios:

- Agregue otro bucle while al principio del script que comprobará si hay un comando quit.
- Agregue una función if al bucle while para verificar **q** o **quit**.

Nota: Asegúrese de incluir todos los niveles requeridos de sangría.

```
while True:
    x=input("Ingrese un número para contar hasta:")
    if x == 'q' o x == 'quit':
        break

    x=int (x)
    y=1
    while True:
        print(y)
        y=y+1
        if y>x:
            break
```

b. Guarde y ejecute el script. La salida debe ser similar a la siguiente, en la que el usuario introdujo dos valores diferentes antes de salir del programa.

```
devasc @labvm: ~/labs/devnet-src/python$ python3 while-loop.py
Introduzca un número para contar: 3
1
2
3
Introduzca un número para contar: 5
1
```

```
2
3
4
5
Introduzca un número para contar: quit
devasc @labvm: ~/labs/devnet-src/python$
```

Parte 7: Métodos de revisión para acceso a archivos.

En esta parte, revisará los métodos para acceder, leer y manipular un archivo.

Paso 1: Cree un programa que lea un archivo externo.

Además de la entrada del usuario, puede acceder a una base de datos, a otro programa informático o a un archivo para proporcionar entrada al programa. La función **open ()** se puede utilizar para acceder a un archivo utilizando la siguiente sintaxis:

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

El parámetro name es el nombre del archivo que se va a abrir. Si el archivo está en un directorio diferente del script, también deberá proporcionar información de ruta. Para nuestros propósitos, solo estamos interesados en tres parámetros de modo:

- **r** - leer el archivo(modo predeterminado si se omite el modo).
- **w** - escribir sobre el archivo, remplazando el contenido del archivo.
- **a** - adjuntar al archivo.

a. Abra un script en blanco y guárdelo como **file-access.py**.

b. Cree el siguiente programa para leer e imprimir el contenido de **devices.txt** que está en el mismo directorio que su programa. Después de imprimir el contenido del archivo, utilice la función **close ()** para eliminarlo de la memoria del ordenador.

```
file=open ("devices.txt", "r")
for item in file:
print(item)
file.close()
```

Nota: El contenido del archivo se establece en un archivo con nombre variable. Sin embargo, esa variable se puede llamar de cualquier forma, dependiendo de lo que el programador elija.

c. Guarde y ejecute su programa. Usted debería de obtener un resultado similar al siguiente.

```
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 Teléfono VoIP

Cisco 3860 Catalyst Switch
devasc @labvm: ~/labs/devnet-src/python$
```

Paso 2: Elimine las líneas en blanco de la salida.

Es posible que haya notado que Python agregó una línea en blanco después de cada entrada. Podemos eliminar esta línea en blanco usando el método **strip ()**.

a. Edite su programa **file-access.py** para incluir el método **strip ()**.

```
file=open ("devices.txt", "r")
for item in file:
    item=item.strip ()
    print(item)
file.close()
```

b. Guarde y ejecute su programa. Usted debería de obtener un resultado similar al siguiente.

```
devasc @labvm: ~/labs/devnet-src/python$ python3 file-access.py
Cisco 819 Router
Cisco 881 Router
Cisco 888 Router
Cisco 1100 Router
Router Cisco 4321
Cisco 4331 Router
Router Cisco 4351
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$
```

Paso 3: Copie el contenido de un archivo en una variable de lista.

La mayoría de las veces, cuando los programadores acceden a un recurso externo como una base de datos o un archivo, quieren copiar ese contenido en una variable local a la que se puede hacer referencia y manipular sin afectar el recurso original.

El archivo **devices.txt** es una lista de dispositivos Cisco que se pueden copiar fácilmente en una lista de Python siguiendo los pasos siguientes:

- Crear una lista vacía.
- Utilice el parámetro **append** para copiar el contenido del archivo en la nueva lista.
- Imprima la lista.

a. Modifique su file-access.py como se muestra en el siguiente ejemplo:

```
devices=[]
file=open («devices.txt», "r»)
for item in file:
item=item.strip ()
dispositivos.append (item)
file.close()
print(devices)
```

b. Guarde y ejecute su programa. Usted debería de obtener un resultado similar al siguiente.

```
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$
```

Paso 4: Desafío: Cree un script para permitir al usuario agregar dispositivos.

¿Qué pasa si desea agregar más dispositivos al archivo **devices.txt** ? Puede crear un programa para abrir el archivo en modo anexado y luego pedir al usuario que proporcione el nombre de los nuevos dispositivos. Complete los siguientes pasos para crear un script:

1) Abra un nuevo archivo y guárdelo como **file-access-input.py**.
2) Para la función **open ()** use el modo **a**, que le permitirá agregar un elemento al archivo **devices.txt**.
3) Dentro de un bucle **while True:**, incruste un comando de función **input ()** que pide al usuario el nuevo dispositivo.
4) Establezca el valor de la entrada del usuario en una variable llamada **NewItem**.
5) Utilice una instrucción **if** que rompa el bucle si el usuario escribe **quit** e imprime la declaración "Todo hecho!".
6) Utilice el comando **file.write (newitem + "\ n")** para agregar el nuevo dispositivo proporcionado por el usuario.
7) Cierre el archivo para quitarlo de la memoria del equipo.

Ejecute y solucione los problemas del script hasta que obtenga resultados similares a los siguientes.

```
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$
```

```
file = open ("devices.txt", "a")
while True:
newItem = input("Enter device name: ")
if Newitem == "exit":
print("All done!")
break
file.write(newItem + "\n")
file.close()
```