## Lab 03:  Python Programming Overview

### Case Study

NetWave Solutions, a fast-growing enterprise in the network infrastructure industry, was experiencing operational inefficiencies due to its outdated manual method for tracking network devices. With a wide array of routers, switches, and security appliances being deployed across various locations, their reliance on manually updated text files made it difficult to maintain consistent and error-free records. As device inventories expanded, the lack of a structured system created confusion, delayed project timelines, and raised concerns about data accuracy.

### Business Challenge

The primary challenge NetWave Solutions faced was streamlining the way devices were logged, tracked, and updated by the IT team. Their manual approach led to duplicated entries, formatting inconsistencies, and time-consuming verification. Furthermore, the process required technical familiarity, which limited who could update records and increased dependency on a small number of staff members. The company needed a simple, scalable solution that could reduce human error and be adopted quickly across teams with varying levels of technical skill.

### Solution

To address operational inefficiencies, NetWave Solutions implemented a Python-based automation tool to streamline device record management. The manual system was replaced with a user-friendly interface that accepted structured input and updated a centralized file, reducing errors and reliance on technical staff.

The solution included data validation, formatting, and cleanup mechanisms to ensure consistency and accuracy. This enabled team members across departments to manage records independently, improving efficiency, collaboration, and scalability for future automation efforts.

1. Start Python and VS Code
2. Explore and review Data Types and Variables
3. Review Lists and Dictionaries
4. Review the Input Function
5. Review how to create and manipulate lists and dictionaries
6. Review If, For, and While Functions
7. Review Methods for File Access

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **//** **Start Python and VS Code**  1. Launch the DEVASC VM.Double-click on the **Terminal** icon to open it.    2. Open the Terminal and type the command **python3 -V.** This verifies Python is installed. You should see output like **Python 3.8.2**.    3. Type the command **python3** to start the Python interpreter, and after entering the command, you will see the Python version information    4. Python can perform basic math operations directly. Use the following syntax for each type of operation:   |  |  |  | | --- | --- | --- | | **Operation** | **Math Example** | **Python Syntax** | | Addition | a + b | a + b | | Subtraction | a-b | a-b | | Multiplication | a × b | a \* b | | Division | a ÷ b | a / b | | Exponentiation | a^b | a \*\* b |   Type the following commands to test each arithmetic operation:    5. A string is any sequence of characters, letters, numbers, symbols, or punctuation marks. Strings must be enclosed in either single quotes **(' ')** or double quotes **(" ")** to be recognized by Python.  When you enter a string in the interactive interpreter, Python will echo it back as the following output:    6. To display a string without the enclosing quotes, use the **print()** function:    7. When you are finished, type **quit()** to exit the interpreter and return to the Linux terminal:    8. Double-click the **VS Code** icon on your DEVASC VM to open it    9. Go to the top menu and click **File > New File**.    10. In the new file, type the following Python command:    11. Click **File > Save As.**    12. Click on **labs.**    13. Click on **devnet-src.**    14. Click on **python.**    15. Save the file as **hello-world.py** inside the path: **labs/devnet-src/python.** Make sure the file extension is **.py**.    16. After pressing the **Save button,** the file name (hello-world.py) appears. This indicates that the file has been successfully saved.    17. You can also run the script using a standalone terminal outside of VS Code. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 18. Now, click on the **Terminal** and then, click on the **New Terminal**.    19. A terminal will launch within VS Code and execute the script.    20. To access your Python lab files enter command: **cd labs/devnet-src/python.**    21. Now you are inside the correct directory to create, view, or run your Python scripts, such as hello-world.py. From here, you can run your script **python3 hello-world.py.**  **Hello World** is the output printed by your Python script.    **//Review Basic Python Data Types and Variables**  1. In this step, you will learn how to identify and understand Python’s core data types using the interactive interpreter.  A data type tells Python what kind of value a variable holds, such as a number, text, or True/False condition. Python is dynamically typed, meaning it figures out the data type when the value is assigned or entered.  We will review these four fundamental data types:   * **Integer (int)**: Whole numbers without decimals * **Float (float)**: Numbers with decimals * **String (str)**: Text or characters * **Boolean (bool)**: Values that are either True or False   Use the **python3 and type()** command to determine the basic data types: int, float, string, and Boolean.    2. Boolean operators are used to compare values and return True or False. Below are examples of how to use them in Python’s interactive interpreter by adding **1<2, 1<1, 1==1, 1<=1**   |  |  | | --- | --- | | **Operator** | **Description** | | > | Greater than | | < | Less than | | == | Equal to | | != | Not equal to | | >= | Greater than or equal to | | <= | Less than or equal to |     3. In Python, variables are used to store data that can be reused in other operations. You assign a value to a variable using the single equal sign **=.** For example, typing **x = 3** creates a variable **x** and assigns it the value **3**. You can then use this variable in both numeric and string operations. For instance, typing **x \* 5** multiplies the value of x (which is 3) by 5, resulting in 15. Likewise, if you type "**Cisco" \* x**, Python repeats the string "Cisco" three times, producing 'CiscoCiscoCisco'. This demonstrates how variables allow you to store values and reuse them in different contexts within the interpreter.    4. Concatenation is the process of joining multiple strings into one. This is useful when combining different pieces of text stored in separate variables. In Python, strings can be concatenated using the + operator or by passing multiple strings as arguments to the **print()** function. First, define three string variables and one space variable:   |  | | --- | | >>> str1="Cisco"  >>> str2="Networking"  >>> str3="Academy"  >>> space=" "  >>> print(str1+space+str2+space+str3) |     5.To print the variables without using a variable to create the space, separate the variables with a comma. Byb using **print(str1,str2,str3)**    6. When working with different data types in Python, you may need to convert (or cast) one type into another. This is especially necessary when combining values like strings and integers in operations such as concatenation. If you try to combine a string with an integer directly using +, Python will return an error:   |  | | --- | | >>> x=3  >>> print("The value of x is " + x) |     7. Use the **str()** function to convert the integer data type to a string data type. by using   |  | | --- | | >>>print ("The value of x is " + str(x))  >>>type(x) |     8. To convert and store the new data type, reassign x as a string:   |  | | --- | | >>>x=str(x)  >>>type(x) |     9. Round a float to a specific number of decimal places. Use an f-strin**g** or the. **format()** method:   |  | | --- | | >>>num = 22/7  >>>f"The value of num is {num}"  >>>pi = "{:.2f}".format(num)  >>>f"The value of pi is {pi}." | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **// Review Lists and Dictionaries**  1. To store multiple ordered items in a single variable, create a list using square brackets **[ ],** placing each item in quotes and separating them with commas. After defining the list, use the **type()** command to confirm it is a list, and **len()** to check how many items it contains. Finally, enter the list’s variable name to display its full contents. The commands to enter include   |  | | --- | | >>>hostnames=["R1","R2","R3","S1","S2"]  >>>type(hostnames)  >>>len(hostnames)  >>>hostnames |     2. In Python, list elements can be accessed and changed using index positions. The first element is at **index 0**, the second at **1**, and so on. You can access the last item using **index -1**. To update a value, assign a new value to a specific index. To remove an item, use the del command followed by the index. The commands to enter include:   |  | | --- | | >>>hostnames[0]  >>>hostnames[-1]  >>>hostnames[0]="RTR1"  >>>hostnames  >>>del hostnames[3]  >>>hostnames |     3. In Python, dictionaries are used to store unordered collections of key/value pairs. Each item in a dictionary consists of a key and a corresponding value. To create a dictionary, use curly braces **{}** and place each key and its value inside, separated by a **colon:**. Multiple key/value pairs are separated by commas, and both keys and values should be enclosed in quotes if they are strings**.**   |  | | --- | | >>>ipAddress={"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}  >>>type(ipAddress) |     4. Unlike lists, the elements in a dictionary cannot be accessed by their position or sequence number. Instead, dictionary elements are referenced using their unique keys. To retrieve or assign a value, you place the key inside square brackets **[ ],** using either single or double quotes for string keys.   |  | | --- | | >>>ipAddress  >>>ipAddress['R1']  >>>ipAddress["S1"] ="10.1.1.10"  >>>ipAddress |     5. Values in a key/value pair can be any other data type, including lists and dictionaries. For example, if R3 has more than one IP address, how would you represent that inside the ipAddress dictionary? Create a list for the value of the R3 key.   |  | | --- | | >>>ipAddress["R3"]=["10.3.3.1","10.3.3.2","10.3.3.3"]  >>>ipAddress | |

|  |  |  |
| --- | --- | --- |
| **//Review the Input Function**  1. Programs often need input from users. The **input()** function allows you to prompt for keyboard input, pausing the program until the user responds and presses **Enter**. You can assign this input to a variable and use it later in the program.   |  | | --- | | >>>firstName = input("What is your first name? ")  What is your first name? User\_Name  >>>print("Hello " + firstName +"!") |     2. To begin, open Visual Studio Code and create a new file. In the script, write Python code that prompts the user to enter their first name, last name, location, and age using the **input()** function. After collecting the inputs, use a **print()** statement to display a well-formatted sentence that includes all the provided information. Write the given code   |  | | --- | | fN = input ("What is your first name?")  lN = input ("What is your last name?")  loc = input("What is your location? ")  age = input("What is your age? ")  print("Hi " + fN, lN + "! Your location is " + loc + " and you are " + age + " years old.") |     3. To save your Python script in Visual Studio Code, start by clicking on the **File** menu at the top left of the window, then select **Save As**. This will open a dialog box where you can choose the location and name of your file. In the file navigation window, browse to the directory **~/labs/devnet-src/python**. In the Name field, type personal-info.py, ensuring that the .py extension is included to indicate it is a Python file. Once everything is set, click the Save button in the bottom right corner to create and save your script in the desired location.    4. To run your saved Python script, open the terminal in your DEVASC VM and type the command **python3 /home/devasc/labs/devnet-src/python/personal-info.py**, then press Enter. This will execute the script and prompt you to input your first name, last name, location, and age. |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **// Review If, For, and While Functions**  1. To review the use of **if** and **else** statements in Python, start by creating a new script named **if-vlan.py** using Visual Studio Code. In this script, define two variables: **nativeVLAN=1** and **dataVLAN=100**. Use an **if/else** conditional statement to compare their values. If both VLANs are equal, the script should print a message indicating they are the same; otherwise, it should print that they are different. Remember that Python requires proper indentation (typically four spaces) to define code blocks under if and else. Once you have typed the script, save and run it.   |  | | --- | | 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.") |     2. Save the script as **vlan-if.py**.    3. Now, **Run** it in the terminal. The output should look like the following.    4. Modify the variables so that nativeVLAN and dataVLAN have the same value which is **nativeVLAN=1** and **dataVLAN=1**. **Save** and **run** the script again.   |  | | --- | | nativeVLAN = 1  dataVLAN = 1  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.") |   5. The output should look like the following.    6. When your program needs to evaluate more than two possible conditions, you can use **elif (else if)** statements between an initial if and a final else. Each elif is checked only if the if condition is false, and once a match is found, Python will execute that block and ignore the rest, including the else. You can include as many elif branches as needed, but only the first matching one will run.   |  | | --- | | 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.") |     8. Save the file as **if-acl.py.**    9. Run **python3 /home/devasc/labs/devnet-src/python/if-acl.py** multiple times to test each statement.    10. Create a **for loop** in Python, which is commonly used to iterate through the elements of a list or to perform a repeated action across a sequence of values. To explore how a for loop works, open the interactive Python interpreter and type in the loop statement. You can use any variable name you like for the loop element; item is often used, though many programmers prefer to shorten it to just i. When entering the code, remember to indent the **print()** function using four spaces (not a tab), as proper indentation is crucial in Python. After writing the for loop and its indented line, press the Enter key twice to execute and exit the loop.   |  | | --- | | >>> devices=["R1","R2","R3","S1","S2"]  >>> for item in devices:  ... print(item) |     11. To display only the items that start with the letter R, you can embed an if statement inside a for loop. This allows Python to evaluate each item in a list and perform an action, like printing, only when a condition is met. To try this out, open the Python interactive interpreter and type the following code carefully.  Make sure the if statement is indented with four spaces, and the **print()** function inside the if block is indented with eight spaces. This nesting ensures Python understands the structure. After writing the loop, press Enter twice to run and exit the block.   |  | | --- | | For item in devices:  ... if "R" in item:  ... print(item) |     12. You can combine a for loop with an if statement to build a new list based on certain conditions. In this example, you create an empty list called switches, then loop through another list called devices. During each iteration, the script checks whether the current item contains the letter "S". If it does, the **append()** method adds that item to the switches list.  To try this, follow the indentation carefully—use four spaces for the if statement and eight spaces for the **append()** method. Here is how it looks in the Python interactive interpreter:   |  | | --- | | >>> switches=[]  >>> for item in devices:  ... if "S" in item:  ... switches.append(item)  ...  >>> switches |     13. Use a while loop to repeatedly execute a block of code as long as a specified condition is true. This differs from an if statement, which only executes once if the condition is met. However, it is important to include an exit condition; otherwise, the loop can run endlessly. Open a blank script file and save it as while-loop.py. Add the following code:   |  | | --- | | x = input("Enter a number to count to: ")  x = int(x)  y = 1  while y <= x:  print(y)  y = y + 1 |     11. Save the file as a **while-loop.py.**    12. Run **python3 /home/devasc/labs/devnet-src/python/while-loop.py**    13. In this modified version of the **while-loop.py** script, instead of using while y <= x, you use an infinite loop with while True: and control when it stops using a break statement. This approach is helpful when you want more flexible control over loop termination.   |  | | --- | | x=input("Enter a number to count to: ")  x=int(x)  y=1  while True:  print(y)  y=y+1  if y>x:  break |     14. Run **python3 /home/devasc/labs/devnet-src/python/while-loop.py**    15. To allow your program to run repeatedly until the user decides to quit, you can embed the number-counting logic inside a while True loop that checks for a quit command. This gives the user the flexibility to enter numbers multiple times or exit the program by typing **q** or quit.   |  | | --- | | 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 |     16. Run **python3 /home/devasc/labs/devnet-src/python/while-loop.py** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **//Review Methods for File Access**  1. Open a blank script and save it as **file-access.py.**    2. Save the file as a **file-access.py.**    3. Add this code in vs code.   |  | | --- | | file = open('/home/devasc/labs/devnet-src/python/devices.txt', "r")  for item in file:  print(item.strip())  file.close() |     4. In terminal, run **cd /home /devasc/labs/devnet-src/python and python3 file-access.py**    5. Python added a blank line after each entry. We can remove this blank line using the strip() method. Edit your file-access.py program to include the strip() method.   |  | | --- | | devices = []  file = open("/home/devasc/labs/devnet-src/python/devices.txt", "r")  for item in file:  item = item.strip()  devices.append(item)  print(item)  file.close()  print(devices) |     6. In Terminal, run **python3 file-access.py**    7. To add more devices to the **devices.txt** file, you can create a Python script that continuously prompts the user for device names and appends them to the file. The loop continues until the user types exit, at which point the program stops and confirms completion.   |  | | --- | | devices = []  file = open("/home/devasc/labs/devnet-src/python/devices.txt", "r")  for item in file:  item = item.strip()  devices.append(item)  print(item)  file.close()  print(devices) |     8. Run **python3 file-access.py**    9. To allow users to add new devices to a file named devices.txt, you can create a Python script called **file-access-input.py**. This script opens the file in append mode using the **open()** function with mode "a", which enables adding content to the end of the file without overwriting existing data. Inside a while True loop, the script repeatedly prompts the user to enter a device name using the **input()** function. The user's response is stored in a variable named newItem. If the user types "exit", the loop breaks and the message **"All done!"** is printed. Otherwise, the device name is written to the file with a newline character using the file.write(newItem + "\n"). Finally, the file is closed using **file.close()** to ensure proper resource management.   |  | | --- | | file = open('/home/devasc/labs/devnet-src/python/devices.txt', "a")  while True:  newItem = input("Enter device name: ")  if newItem == "exit":  print("All done!")  break  file.write(newItem + "\n")  file.close() |     10. Run **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 | |