## Lab 23-01: Programming with Python

### Scenario

You are going to perform a security assessment of a company’s network using pen test automation through Python. You have been asked to identify any potential vulnerabilities that attackers could exploit. Before performing the pen test automation using Python, you have decided to go through the basic Python codes to help you recall the commands.

### Solution

To perform Python codes, you need a comprehensive tool that can help you execute the commands. You can use Python Software (Python 3.11), or you can also use Visual Studio Code (VS Code) and Pycharm to run Python commands.

|  |
| --- |
| **// Print**  1. In Python, the print statement instructs the program to display text or numbers on the screen. This is a line of code that instructs Python to display the text **"Hello World!"**    **//Variables for Strings**  A variable is created this way:  **name = "IPS"** , the variable name refers to the text string "IPS". Note that it was my choice to call it **name**. I could have called it **my\_name, xyz**, or something else. It's up to me how to name my variables, within limits. With the string **"IPS"** assigned to the variable **name**, my Python code does not have to specify **"IPS"** again. Whenever Python encounters **name**, Python knows that it is a variable that refers to "IPS".  For example, if you write,  **name = "IPS"**  **print(name)**  Python displays **IPS**    **//Variables for Numbers**  A string isn't the only thing you can assign to a variable. You can also assign a number.  **my\_favourite\_number = 18**  **print (my\_favourite\_number)**    Having coded the statement above, whenever you write **my\_favourite\_number** in your code, Python knows you mean **18.**  **//Math Expressions: Familiar Operators**  1. In Python, the programming term for a calculation is math expression. Familiar operators in math expressions are + (add), - (subtract), \* (multiply), and / (divide).  2. To perform addition, you can use the commands;  **first=5**  **second=10**  **code = first+second**  **print (code)**    3. To perform subtraction, you can use the commands;  **first=10**  **second=5**  **code = first-second**  **print (code)**    4. To perform division, you can use the commands;  **Print (10/3)**    **// Math Expressions: Unfamiliar Operators**  1. **%** is the Modulo operator. It divides a number by another number, but it withholds the division's outcome. Once the first number is divided by the second number, you are given the remainder. The remainder of 10 divided by 3 equals 1.  2. \*\* is the Exponential operator. It provides the outcome of raising the first operand to the second operand's power.    **//Comparison operators**  **1. Equal to (==)**: This operator checks if the operands on both sides are equal. In this case, 2 == 3 evaluates to False because 2 is not equal to 3.  **2. Not equal to (! =)**: This operator checks if the operands on both sides are not equal. In this case, 2! = 3 evaluates to True because 2 is not equal to 3.  **3. Less than (<)**: This operator checks if the operand on the left is less than the operand on the right. In this case, 2 < 3 evaluates to True because 2 is less than 3.  **4. Less than or equal to (<=)**: This operator checks if the operand on the left is less than or equal to the operand on the right. In this case, 2 <= 3 evaluates to True because 2 is less than 3.  **5. Greater than (>)**: This operator checks if the operand on the left is greater than the operand on the right. In this case, 2 > 3 evaluates to False because 2 is not greater than 3.  **6. Greater than or equal to (>=)**: This operator checks if the operand on the left is greater than or equal to the operand on the right. In this case, 2 >= 3 evaluates to False because 2 is not greater than 3.    **// Logical Operators**  1. The first line **print(True and False)** shows the **AND** operator. In Python, the **and** operator returns True only if both operands are True. In this case, since one operand is False, the entire expression evaluates to **False**.  2. The second line **print(True or False)** shows the **OR**operator. In Python, the **or**operator returns **True** if at least one operand is **True**. Since one operand is **True,** the entire expression evaluates to **True**.  3. The third line **print(not True)** shows the **NOT** operator. In Python, the **not** operator reverses the logical state of its operand. In this case, since the operand is **True**, the not operator returns **False**.    **//Lists**  1. First, assign some string values to some variables.    2. The definition of a list begins the same way the definition of any variable begins, in this case, **cities =** But when you are defining a list, you enclose everything to the right of the equal sign in square brackets:  **cities = ["Atlanta," "Baltimore," "Chicago," "Denver," "Los Angeles," "Seattle"]**    3. Since Denver is the fourth element in the list (remember, the numbering starts at 0, so the fourth element has an index of 3), Python displays **Welcome to Denver**.  The list contains six elements, "Atlanta"—cities[0]—through "Seattle"— cities[5]. For example, you want to add a seventh city, New York. The statement begins with the list name. Next there is a dot, then the keyword append. The value, in this case the string "New York," is enclosed in parentheses:  **cities.append("New York")**    4. If you want to **insert** "**New York**" at the beginning of the list. The statement begins with the list name, followed by a **dot, and** then comes the keyword **insert;** the rest of it is enclosed in **parentheses.** But this time, there are two things to specify: the **index** that tells Python where you want the element placed following a **comma** and **space** the value of the element. The statement will be:  **cities.insert(0, "New York")**    **//Tuples**  When you are confident that these will always be the first five numbers, their order will not change. You will never need to replace one of them with another state. You will never need to add another state. And, barring extraordinary events, You will never need to delete one of them. In that case, a tuple is created. Tuple means a list that is written in stone. You code a tuple as you would a list, with one exception:  my\_tuple = ("1", "2", "3", "4","5")  or  my\_tuple=(1, 2, 3, 4, 5)  **# To access the elements of a tuple**  For Output 1, you will write  print(my\_tuple[0])  **#For Output 2, you will write**  print(my\_tuple[1])  **# To iterate over a tuple**  for element in my\_tuple:    print(element)  **# To check if an element exists in a tuple**  if 3 in my\_tuple:    print("3 exists in the tuple")  # To get the length of a tuple  print(len(my\_tuple))    **Output:**    **//Dictionary**  1. Dictionaries in Python are collections of **key-value pairs**. The curly braces are utilized to define a dictionary. Inside the curly braces, each key-value pair is separated by commas. Each key-value pair is enclosed in quotation marks, and a colon separates the key from its value. The purpose of a dictionary is to store information that you can later lay your hands on.  The dictionary in the below example comprises three pairs: key is **"name**", the value is **"John Doe**" key is "**age**", value is **30** key is "**city**", value is "**New York**”. For example, you might want to know what is the age of John is?    2. Iterate over the dictionary’s keys:  The for loop **for key in my\_dict** iterates over the keys in the dictionary **my\_dict**.  Inside the loop, the code block **(print(key))** prints the current key to the console.  Iterate over the dictionary’s values:  The for loop **for value in my\_dict.values()** iterates over the values in the dictionary **my\_dict**. The **.values()** method returns a view object that contains all the values in the dictionary. Inside the loop, the code block **(print(value))** prints the current value to the console.  Iterate over the dictionary’s key-value pairs:  The for loop **for key, value in my\_dict.items()** iterates over the key-value pairs in the dictionary **my\_dict**. The **.items()** method returns a view object that contains all the key-value pairs in the dictionary as tuples.  Inside the loop, the code block **print(f"{key}: {value}")** uses f-strings to format the output, printing the key followed by a colon and then the value.    **OUTPUT:**    3. Add a new key-value pair:  The line **my\_dict["job"] = "Software Engineer"** adds a new key-value pair to the dictionary **my\_dict**. The key is "job" and the value is "Software Engineer".  Print the value of a key:  The line **print(my\_dict["job"])** prints the value associated with the key "job" in the dictionary **my\_dict**.  Remove a key-value pair:  The line **del my\_dict["age"]** removes the key-value pair with the key "age" from the dictionary **my\_dict**. The **del** keyword is used to delete objects in Python.  Check if a key exists:  The line **print("name" in my\_dict)** checks if the key **"name"**exists in the dictionary **my\_dict**. The **in** keyword is used to check for **membership** in Python. It returns **True** if the key is found in the dictionary and **False** otherwise.    **//Functions**  1. The definition begins with the keyword **def (for define);** after that, we write a name **greet** (it can be any legal variable name you like). The function name is followed by parentheses, and the line ends with a colon. If you put some information inside the parentheses, that information is passed to the function. The function can then use the information when it executes.  2. This code defines a function named **greet** that takes one argument, **name**. The function then uses an **f-string** to print a personalized greeting to the console. |