**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **23-May-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **Udemy** | **USN:** | **4AL16EC058** |
| **Topic:** | **Strings** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  In my first session today I have studied about the Strings in Python. **Python Strings :**String Literals: String literals in python are surrounded by either single quotation marks, or double quotation marks.  'hello' is the same as "hello".  You can display a string literal with the print() function:  Example:  print("Hello") print('Hello') Assign String to a Variable Assigning a string to a variable is done with the variable name followed by an equal sign and the string: Example a = "Hello" print(a) Multiline Strings You can assign a multiline string to a variable by using three quotes: Example You can use three double quotes:  a = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.""" print(a) Strings are Arrays Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.  However, Python does not have a character data type, a single character is simply a string with a length of 1.  Square brackets can be used to access elements of the string. Example Get the character at position 1 (remember that the first character has the position 0):  a = "Hello, World!" print(a[1]) Slicing You can return a range of characters by using the slice syntax.  Specify the start index and the end index, separated by a colon, to return a part of the string. Example Get the characters from position 2 to position 5 (not included):  b = "Hello, World!" print(b[2:5]) String Length To get the length of a string, use the len() function. Example The len() function returns the length of a string:  a = "Hello, World!" print(len(a)) String Methods Python has a set of built-in methods that you can use on strings. Example The strip() method removes any whitespace from the beginning or the end:  a = " Hello, World! " print(a.strip()) # returns "Hello, World!" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **23-May-2020** | **Name:** | **Raziya Banu** | |
| **Course:** | **Udemy** | **USN:** | **4AL16EC058** | |
| **Topic:** | **Boolean data types** | **Semester & Section:** | **8th sem & ‘B’ section** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| In my second session today I have studied about the Boolean datatypes of python.  **Boolean datatypes :**  [Booleans](https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not) are a concept that exists in every programming language. A boolean represents the idea of "true" or "false". When you are writing a program, there are often circumstances where you want to execute different code in different situations. Booleans enable our code to do just that.  You can declare a boolean value in your code using the keywords True and False (note the uppercase). The following code would create two boolean values and assign them to variables.  mullet\_looks\_good = False  python\_is\_fun = True  More commonly, a boolean value is returned as a result of some kind of comparison. The following code example would store a boolean value of False in the have\_same\_name variable after using the [equality comparison operator](https://docs.python.org/3/library/stdtypes.html#comparisons), the == symbol.  my\_name = "Wammu"  your\_name = "Kars"  have\_same\_name = my\_name == your\_name Boolean logic Booleans are used in your code to make it behave differently based on current conditions within your program. You can use boolean values and comparisons in conjunction with the if, elif, and else keyoards as one means to achieve this.  my\_age = 10  if my\_age >= 100:  print("One hundred years old! Very impressive.")  elif my\_age <= 3:  print("Awwww. Just a baby.")  else:  print("Ah - a very fine age indeed")  In addition to testing for truth, you can also check if conditions are not true using the not keyword.  favorite\_team = "Vikings"  if not favorite\_team == "Vikings":  print("Oh - how unfortunate.")  else:  print("Skol, Vikings!") More complex boolean logic Sometimes you will need to evaluate multiple conditions in your boolean logic. For this purpose, you'll combine the and and or keywords. The and keyword compares two boolean values and returns True if both are true. The or keyword compares two values and returns True if any of the statements are true.  Let's look at an example. That uses the in keyword to see if a string is inside a **list** of values (we'll cover lists in a future article).  favs = ["Donatello", "Raphael"]  if "Michelangelo" in favs and "Donatello" in favs:  print("Those are my favorite ninja turtles too!")  elif "Michelangelo" in favs or "Donatello" in favs:  print("Well, one out of two isn't bad...")  else:  print("Huh - not what I would have chosen.") | | | |