**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **21-05-2020** | **Name:** | **Dhanya Shetty** |
| **Course:** | **TCSion** | **USN:** | **4AL17EC026** |
| **Topic:** | **Write a winning Resume and Cover letter** | **Semester & Section:** | **6th A** |
| **Github Repository:** | **Dhanya Shetty\_026** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| C:\Users\Hp\Pictures\tcs 21may.PNG |
| C:\Users\Hp\Pictures\tcs and python\20200521_131057.jpg   |  |  | | --- | --- | |  | | |  | | **C:\Users\Hp\Pictures\tcs and python\20200521_131109.jpg** | | |
| |  |  |  | | --- | --- | --- | | **Date: 21-05-2020** |  | **Name: Dhanya Shetty** | | **Course: Python** |  | **USN:4AL17EC026** | | **Topic: The Basics: Functions and conditionals** |  | **Semester & Section:6th A** | |

|  |
| --- |
|  |
|  |
|  |
| **AFTERNOON SESSION DETAILS** | |
| **Image of sessions**  **C:\Users\Hp\Pictures\py01 21may.PNG** | |
| C:\Users\Hp\Pictures\py02 21may.PNGC:\Users\Hp\Pictures\py03 21may.PNG  C:\Users\Hp\Pictures\py04 21may.PNG  Parameters to **functions** are references to objects, which are passed by value. When you pass a **variable** to a **function**, **python** passes the reference to the object to which the **variable** refers (the value). Not the **variable** itself. **Functions** have a local **variable** table called a local namespace.  **Variables in Python:** A variable is essentially a place where we can store the value of something for processing later on. Imagine you wanted to write a program that doubled a number for us, not the most exciting of programs I know but it is a good example. We would first want to define a variable to store our number, double it and then print it out.  **Variables:** Recall that a variable is a label for a location in memory. It can be used to hold a value. In statically typed languages, variables have predetermined types, and a variable can only be used to hold values of that type. In Python, we may reuse the same variable to store values of any type.  A variable is similar to the memory functionality found in most calculators, in that it holds one value which can be retrieved many times, and that storing a new value erases the old. A variable differs from a calculator’s memory in that one can have many variables storing different values, and that each variable is referred to by name.  **Defining variables**  To define a new variable in Python, we simply assign a value to a label. For example, this is how we create a variable called count, which contains an integer value of zero:  count = 0  This is exactly the same syntax as assigning a new value to an existing variable called count. Later in this chapter we will discuss under what circumstances this statement will cause a new variable to be created.  If we try to access the value of a variable which hasn’t been defined anywhere yet, the interpreter will exit with a name error.  We can define several variables in one line, but this is usually considered bad style:  # Define three variables at once:  count, result, total = 0, 0, 0  # This is equivalent to:  count = 0  result = 0  total = 0  Here is an example of variables in different scopes:  # This is a global variable  a = 0  if a == 0:  # This is still a global variable  b = 1  def my\_function(c):  # this is a local variable  d = 3  print(c)  print(d)  # Now we call the function, passing the value 7 as the first and only parameter  my\_function(7)  # a and b still exist  print(a)  print(b)  # c and d don't exist anymore -- these statements will give us name errors!  print(c)  print(d)  **Functions in Python:** So now that we’ve got a basic understanding of variables in Python, let’s now move on to functions. If you’ve never encountered functions before then you can think of them as blocks of instructions, imagine you had a maths program that could either double or triple a number of your choice. You would want to define two distinct functions that would take in a number and either double or triple it.  Function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.  As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*  **Defining a Function**  You can define functions to provide the required functionality. Here are simple rules to define a function in Python.   * Function blocks begin with the keyword **def** followed by the function name and parentheses ( ( ) ). * Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses. * The first statement of a function can be an optional statement - the documentation string of the function or *docstring*. * The code block within every function starts with a colon (:) and is indented. * The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.   Syntax  def functionname( parameters ):  "function\_docstring"  function\_suite  return [expression]  By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.  Example  The following function takes a string as input parameter and prints it on standard screen.  def printme( str ):  "This prints a passed string into this function"  print str  return  **Summary:**  In this section we learnt:   * Define a **function**:  1. def cube\_volume(a): 2. return a \* a \* a  * Write a **conditional**block:  1. message = "hello there" 3. if "hello" in message: 4. print("hi") 5. else: 6. print("I don't understand")  * Write a conditional block of **multiple conditions**:  1. message = "hello there" 3. if "hello" in message: 4. print("hi") 5. elif "hi" in message: 6. print("hi") 7. elif "hey" in message: 8. print("hi") 9. else: 10. print("I don't understand")  * Use the and operator to check if **both conditions** are True at the same time:  1. x = 1 2. y = 1 4. if x == 1 and y==1: 5. print("Yes") 6. else: 7. print("No")   Output is Yes since both x and y are 1.   * Use the or operator to check if **at least one condition** is True:  1. x = 1 2. y = 2 4. if x == 1 or y==2: 5. print("Yes") 6. else: 7. print("No")   Output is Yes since x is 1.   * Check if a value is of a certain **type** with:  1. isinstance("abc", str) 2. isinstance([1, 2, 3], list)   or   1. type("abc") == str 2. type([1, 2, 3]) == lst | |