PYTHON DAY2 ASSIGNMENT

Name: Tanvi Kulkarni

Python Functions

A1: Functions in Python are those that are defined to perform some task or action. They have a definition which includes the name, the code of the task/ action and a return statement which holds the output of the function. We have to call the function in order to use it. They are of two types:

1) User defined functions: Python lets a user define their own custom functions which are declared as follows:
   ```
   def functionName():
           #code defined here to fulfil a task/ action
           return   # output of the function is written here
   functionName() #function call
   ```
2) Built-in functions: Python also has a wide range of functions which are already provided for ease of programming. They are generic in nature and can be used directly in the code.
   ```
   name= “Tanvi”
   print(name.upper()) #all the letters will be uppercased: TANVI
   print(name.lower()) #all the letters will be lowercased: tanvi
   ```
   In the above example, .lower() and .upper() are built in functions.

A2: Simple function in python can be used by following the way of defining user-defined functions in Python (syntax written above in A1). Example given below:

```
#Code for Addition of two numbers: num1, num2

def addTwoNums(num1,num2):

        return (num1+num2) # return addition output

print(addTwoNums(2,3)) #function call returns 5
```

A3: The return statement holds the value which is produced as an output by the function. In python, a function can return any data type value: string, int, float, boolean etc. It is also possible to create a void return type by simply not writing a return statement in the function body.

A4: A parameter is the actual value which is passed to a function as it is needed to perform the action/ task by the function. It can be of any data type. Example given below:

```
def scanNum(num):

        return (num+1)
```

```
scanNum(23)
```

In the above example, we can see that a number ‘num’ is required by the function to compute an arithmetic operation and produce an output. ‘num’ is an argument and 23 is the parameter.

A5: There will be syntactic differences between a function with parameters and no parameters. The function with 0 parameters will have no arguments in its definition and if we try to pass actual parameters to it, it will generate an error. In such functions, we make use of the ‘input’ keyword to take any user-input data required by the function. The function with parameters will take equal number of parameters in the call as many there are in its definition. Any other variation than this, will generate errors. Example given below:

1) No parameters function

```
def demoZeroParam():
        #code here
        a = int(input("Enter the number: "))
        return a
demoZeroParam()
```

2) Parameterized function

```
def demoParams(a,b,c):
    #code here
    return (a+b+c)
demoParams(1,2,3) #passing 3 parameters
```

Python Logging

A6: To track and monitor the code, we can make use of a specialised library in Python called ‘logging.’ It helps us identify any bottlenecks, errors or discrepancies in our code in a systematic manner.

A7: We can use a logger.info(msg) to display that things are working normally or we may use logger.error(msg) to display an error message given by ‘msg’.

A8: The default logging level in Python's logging module is ‘warning’. This means that some unexpected behaviour has occurred, but it is not interfering with the completion of execution of code. It only indicates that problems may occur in the future.

A9: To log a message with an error level in Python, we can use the logging module and call the error() method on the logger which takes a string argument, which contains the actual error message.

A10: The basicConfig() function in Python's logging module is used to configure the default logging behaviour. It sets up the root logger with default settings, and we can use it to specify various parameters, such as the logging level, log format, output destination etc.

Python Exception Handling

A11: Any event that causes an abnormal behaviour, irregular termination or interrupts the normal functioning of a code script in any way can be termed as an exception. These events need to be handled by the programmer.

A12: Many commonly occurring exceptions such as IndexError, ValueError, ArithmeticError etc are already built-in in Python and don't need to be explicitly handled by the programmer. However, Python also support custom-built exceptions to be declared. These need to be defined by the programmer and used appropriately wherever required. In custom built ones, we need to inherit 'Exception' class, define the name and body of the exception and then use it in out code wherever required. Example given below:

```
# Custom Exception definition

class ValuesCustomError(Exception):

        #code for the exception

#Use of the exception defined

def valueTest():

        try:

                a=int(input("Enter the number: "))

        except ValuesCustomError:

                print("ValuesCustomError occurred.")

        else:

                return a

valueTest()
```

A13: Basic syntax for try and except in Python:

```
def addNums():
    try:
        a=int(input("Enter the number: "))
        b=int(input("Enter the number: "))
    except ValueError:
        print("Incorrect values entered")
    else:
    return (a+b)
addNums()
```

A14: If we do not handle exceptions, our code will log a huge list of errors, and this will make it difficult for us to identify the root cause of failures, thereby prolonging the corrective actions. It is also not a good coding practice to not handle exceptions. It can be interpreted that the developers and testers do not follow any standards of coding. Especially if this failure occurs after the software is live and running in the production environment, the client may not react kindly to it, which may have deprecating results.

A15: The 'try' block includes those steps which may raise an exception. The 'except' block holds the code to handle the exception, if it arises. The 'else' block is used to execute the part of code when no exception occurs. The 'finally' block is the code that always runs irrespective of the occurrence of an exception.

Python Regular Expressions

A16: Python has a built-in library called 're' which stands for Regular Expressions. It is used to specify search patterns for strings.

A17: We can specify the string which needs to be searched and provide the pattern based on which we need to search the string. Then we can use the .search() function from 're' library in python. Example given below:

```
pattern="pattern-here"
string="the-string-to-be-searched-here"
matchedExp=re.search(pattern,string)
```

A18: The .search() function will look for the pattern in the whole string and the .match() function will look for the pattern only in the beginning of the string.

A19: '\d' is used to match the characters in a string for digits (0 to 9).

A20: The .sub(pattern,replacement,string) is used to replace occurrences of the 'pattern' with the 'replacement' in the provided 'string'.

Python Modules and Packages

A21: A module is a group of functions in Python. Example given below:

```
class ArithmeticCalc:
        def Addition(a,b):
                #code
        def Subtraction(a,b):
                #code
        def Multiplication(a,b):
                #code
        def Division(a,b):
                #code
```

The class 'ArithmeticCalc' is a module which contains functions for simple operations.

A22: In Python, we can import a module by mentioning it at the top of the program in which we want to use the functions of module. Example given below:

```
#Importing the above mentioned 'ArithmeticCalc' module from A21
import ArithmeticCalc as ac
ac.Addition(2,3)
ac.Subtraction(6,4)
```

A23: In a package, we can have multiple modules, and each module may have multiple functions. When we use 'import ABC' we import all the modules available within the package 'ABC'. When we use 'from ABC import z", we only want to, specifically want to import the module 'z' from the package 'ABC'. Example given below:

```
import numpy as np
from math import pi
```

A24: In Python, we can view the documentation of a particular module by using 'help' as follows: help(moduleName). In order to view the functions within a module, we can use 'dir' as follows: dir(moduleName).

A25: In a package, we can have multiple modules, and each module may have multiple functions. Example given below:

```
class DemoPackage:
```

```
class Module1:
        # code for different functions
        def method1():
                #code for method
        def method2():
                #code for method
class Module2:
        # code for different functions
        def method1():
                #code for method
        def method2():
                #code for method
```

In the above example, ‘DemoPackage’ is the pacakage and it contains many modules like ‘Module1’ and ‘Module2’ and each of those modules, contain multiple functions like ‘method1’ and ‘method2’.