Python Notes - Week 3

Date:- 02/03/2022

**Python Functions**- Functions are something which we use to put some commonly done tasks together so that instead of writing the same code again and again for different inputs, what we can do is call the function to use the same code only over and over again. It also helps in breaking our program into smaller and modular chunks as our program gets bigger and more complex. In Python, we have some built-in functions that we can use for our convenience and also we can create our own functions. In order to create a function in Python, we make use of the 'def' keyword. After specifying the function header, we can specify an optional docstring which describes what this particular function does. Example:-

| def pyFunction():  print("My first python function.")   pyFunction() # Function Call |
| --- |

-> **return statement**- The function return statement is used to exit from a function and go back to the place from where it was called along with the value returned by the function. The return statement can consist of a variable, an expression, or a constant and if none of them is present, the return statement will simply return a 'None' object. Example:-

| def square\_value(num\_val):  return num\_val \*\* 2   print(square\_value(7)) print(square\_value(5)) |
| --- |

Some common built-in functions in python include:-

-> abs()- returns absolute value of a number

-> bool()- Converts a Value to Boolean

-> complex() - Creates a Complex Number

-> dict() - Creates a Dictionary

-> type() - Returns the type of the object

-> sum() - Adds items of an Iterable

-> str() - returns the string version of the object

-> print() - Prints the Given Object

-> range() - return sequence of integers between start and stop

**Python Function Arguments**

-> **Default Arguments**- A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. We can provide a default value to an argument by using the assignment operator (=). Another important thing to remember is always specify default arguments after non-default arguments otherwise it will throw an error. Example:-

| def personInfo(name, age=29):  print("Name: ", name)  print("Age: ", age)    personInfo("Johnny") |
| --- |

-> **Keyword Arguments**- In Keyword arguments, we specify the name of the argument along with the value, so that there is no restriction regarding the order of parameters. Also we can mix positional arguments with keyword arguments during a function call, but we need to ensure that keyword arguments must follow positional arguments otherwise function call will result in an error. Example-

| def student(name, gender):  print(name, gender)     # Keyword arguments student(name='James', gender='Male') student(gender='Female', name='Helen') student('Male', name='George') |
| --- |

-> **Arbitrary Arguments**- These types of arguments are used when we are not certain about the number of arguments that will be passed into a function. We can deal with such kind of a situation by using two special symbols:-

a) **\*args**(For accepting any number of non-keyword arguments)- The arguments passed get wrapped up into a tuple before being passed into the function. Inside the function, we use a for loop to retrieve all the arguments back. Example:-

| def personName(\*args):  for arg in args:  print (f"Hello {arg}")   personName('Nakul', 'Rohan', 'Kabir', 'Rohit') |
| --- |

b) **\*\*kwargs**(For accepting any number of keyword arguments)- The arguments passed get wrapped up inside a directory before being passed into the function. Inside the function, we use a for loop to retrieve all the arguments back. Example:-

| def items(\*\*kwargs):  for key, value in kwargs.items():  print (f"{key} : {value}")   items(fruit ='Mango', vegetable = 'Capsicum') |
| --- |

**Lambda Functions**:- Lambda Function, also referred to as ‘Anonymous function’ is the same as a regular python function but can be defined without a name. While normal functions are defined using the def keyword, anonymous functions are defined using the lambda keyword. However,they are restricted to single line of expression. They can take in multiple parameters as in regular functions.

Syntax for lambda function is given by- lambda arguments: expression. Example:-

| num\_cube = lambda x : x\*x\*x print(num\_cube(6)) |
| --- |

**Control Flow Statements and Loops**

Conditional Statements- In Python, conditional statements act depending on whether a given condition is true or false. We can execute different blocks of codes depending on the outcome of a condition. Conditional statements always evaluate to either True or False.

There are three types of conditional statements.

1) **if statement**- It takes a condition and evaluates to either True or False. If the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and the control moves to the next line. Example:-

| num\_val = 6 if num\_val > 5:  print(num\_val \* num\_val) # Calculate square of num\_val |
| --- |

2) **if-else**- The if-else statement checks the condition and executes the if block of code when the condition is True, and if the condition is False, it will execute the else block of code. Example:-

| name = input('Enter a name: ')  if name == "Anish":  print("Name is valid!") else:  print("Invalid Name!") |
| --- |

3) **if-elif-else-** In Python, the if-elif-else condition statement has an elif block to connect multiple conditions one after another. The elif statement checks multiple conditions one by one and if the condition fulfills, then executes that code. Example:-

| def user\_check(choice):  if choice == 1:  print("Admin")  elif choice == 2:  print("Editor")  elif choice == 3:  print("Guest")  else:  print("Wrong entry")  user\_check(1) user\_check(2) user\_check(3) |
| --- |

**Looping Statements**

1) **For loop**- By using a for loop, we can iterate any sequence or iterable variable. The sequence can be string, list, dictionary, set, or tuple. Example:

| num\_list = [16, 5, 3, 8, 14, 21, 5, 4, 11]  for val in num\_list: # val is the variable that holds the value of the item inside the sequence on each iteration.  print(val) |
| --- |

-> **range function**- Using this function, We can generate a sequence of numbers. For example, range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start, stop,step\_size). step\_size defaults to 1 if not provided.

-> **for loop with else**- In python, we can have an optional else block with our for loop. The else part is executed if the items in the sequence used in for loop exhausts. We can also use break keyword to stop a for loop but in that case, the else part will get ignored.

a) **Break Statement**- It is used when we want to terminate the loop as soon as the condition is fulfilled instead of doing the remaining iterations. Whenever a break statement is encountered, the loop immediately terminates and the control flows to the statement mentioned after the body of the loop.

b) **Continue Statement**- The continue statement is used to skip the current iteration and continue with the next iteration.

c) **Pass Statement**- A pass statement is a Python null statement. When the interpreter finds a pass statement in the program, it returns no operation. Nothing happens when the pass statement is executed. Example:-

| months = ['January', 'June', 'March', 'April'] for mon in months:  pass print(months) |
| --- |

| for i in range(2, 10, 2):  print(i) else:  print("No items left.") |
| --- |

d) **Match Statements**- These statements are somewhat similar to switch statements, and were released in the latest Python version 3.10. The case clause consists of a pattern to be matched to the variable, a condition to be evaluated if the pattern matches, and a set of statements to be executed if the pattern matches. We can write multiple case statements for multiple possibilities for a given variable. Each case statement has a pattern that has to be matched. Example:

| bool\_val = False match bool\_val:  case True:  print("Incorrect Assumption!")  case False:  print("Correct Assumption!") |
| --- |

**Classes**- A class is a collection of objects or a blueprint of objects defining the common attributes and behavior. Well, it logically groups the data in such a way that code reusability becomes easy. A class is defined using a “Class” Keyword. The attributes are data members (class variables and instance variables) and methods which are accessed via dot notation. Syntax of class is:-

| class ClassName:  # Statement-1  .  .  .  # Statement-N |
| --- |

-> **Class Objects-** They are an instance of a class. It is an entity that has some state and behavior. In simple terms, it is an instance of a class that can access the data. To create an object, use the following code:-

Syntax: obj = className() # obj is the name of the object. Example:-

| class Car:   color = "light gray"   def \_\_init\_\_(self,model\_name, year): # Constructor Method  self.model\_name = model\_name   self.year = year   def displayInfo(self): # User-defined method  print(self.model\_name,self.year)    c1 = Car("Toyota", 2016) # Object Creation print(c1.color) c1.displayInfo() |
| --- |

-> **Class variable-** It is a variable that is shared by all the different objects/instances of a class and it's value is assigned in the class itself only.

-> **Instance variables** are variables which are unique to each instance. Instance variables are variables whose value is assigned inside a constructor or method with the help of 'self' keyword. Example:-

| class Shark:  animal = 'shark' # Class Variable     def \_\_init\_\_(self, type, color): # Constructor Method   self.type = type # Instance Variables   self.color = color    animal\_1 = Shark("aquatic", "blue") print('Animal details:')  print(animal\_1.animal) print(animal\_1.type) print(animal\_1.color) |
| --- |

**Practice Programs**

Q.1 Python program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Sol:-

| for val in range(51):  if val % 3 == 0 and val % 5 == 0:  print("FizzBuzz")  elif val % 3 == 0:  print("Fizz")  elif val % 5 == 0:  print("Buzz")  else:  print(val) |
| --- |

Output:

| FizzBuzz 1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17 Fizz 19 Buzz Fizz 22 23 Fizz Buzz 26 Fizz 28 29 FizzBuzz 31 32 Fizz 34 Buzz Fizz 37 38 Fizz Buzz 41 Fizz 43 44 FizzBuzz 46 47 Fizz 49 Buzz |
| --- |

Q.2 Python program to find numbers between 100 and 400 (both included) where each digit of a number is an even number.

Sol:-

| even\_items = [] for val in range(100, 401):  s = str(val)   if (int(s[0]) % 2 == 0) and (int(s[1]) % 2 == 0) and (int(s[2]) % 2 == 0):  even\_items.append(s)  print(",".join(even\_items)) |
| --- |

Output:- 200,202,204,206,208,220,222,224,226,228,240,242,244,246,248,260,262,264,266,268,280,282,284,286,288,400

Q.3 Define a Python function student(). Using function attributes display the names of all arguments.

Sol:-

| def student(student\_id, student\_name, student\_class):  return f'Student ID: {student\_id}\nStudent Name: {student\_name}\nClass: {student\_class}'   print(student('09171', 'Ben Adams', 'XII')) |
| --- |

Output:-

| Student ID: 09171 Student Name: Ben Adams Class: XII |
| --- |

Q.4 Python program to create two empty classes, Student and Marks. Now create some instances and check whether they are instances of the said classes or not. Also, check whether the said classes are subclasses of the built-in object class or not.

Sol:-

| class Student:  pass  class Marks:  pass  student\_1 = Student() marks\_1 = Marks() print("Instance or Not?", isinstance(student\_1, Student)) print("Instance or Not?", isinstance(marks\_1, Student)) print("Instance or Not?", isinstance(marks\_1, Marks))  print("Subclass or Not?", issubclass(Student, object)) print("Subclass or Not?", issubclass(Marks, object)) |
| --- |

Output:-

| Instance or Not? True Instance or Not? False Instance or Not? True Subclass or Not? True Subclass or Not? True |
| --- |

Q.5 Python function that accepts a string and calculates the number of uppercase letters and lowercase letters.

Sol:-

| str\_val = input("Enter a string: ") upper\_letter = lower\_letter = 0  for val in str\_val:  if val.isupper():   upper\_letter += 1  elif val.islower():  lower\_letter += 1  else:  pass  print(f"No of uppercase letters are {upper\_letter}") print(f"No of lowercase letters are {lower\_letter}") |
| --- |

Output:

| Enter a string: a QUicK brOwN fOX No of uppercase letters are 7 No of lowercase letters are 7 |
| --- |