Python Notes - Week 3

Date:- 01/03/2022

**Data Types In Python**

Data types basically refers to the kind of value stored in a variable on the basis of which operations are performed on that data. Also in Python, everything is considered to be an object so data types are actually “classes” and variables are “instances” (object) of these classes.

1. **Numeric Data Types:-** It represents the data which has some numerical value. Numeric values can be integer(int), floating number(float) and complex numbers(complex).

* Integers- The value of integers is represented by the “int” class. It contains positive or negative whole numbers (without fraction or decimal).
* Float- The value of floating-point numbers is represented by the “float” class. It is a real number with floating point representation. It is specified by a decimal point.
* Complex Numbers – Complex numbers are represented with the help of a “complex” class. It is specified as *(real part) + (imaginary part)j*. For example – 11 + 9j
* In order to determine the type of value a variable holds or to which particular class a variable belongs to we use the “type()” function.

Example:

| int\_var = 54  float\_var = 87.43  complex\_var = 4 + 2j  print("Type of ", int\_var, " is ", type(int\_var))  print("Type of ", float\_var, " is ", type(float\_var))  print("Type of ", complex\_var, " is ", type(complex\_var)) |
| --- |

Output of above program:-

| Type of 54 is <class 'int'> Type of 87.43 is <class 'float'> Type of 4 + 2j is <class 'complex'> |
| --- |

1. **String Data Type:-** A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by the “str” class. Example:- str\_1 = “My name is Robert Johnson”, str\_2 = ‘How are you?’’

To access individual elements of a string, indexing method is used Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on. Some common string methods include:-

* print (len(String\_Name)) - Displays the length of string
* print (String\_Name.index(“Char”)) - Locate a character in String
* print (String\_Name.count(“Char”)) - Count the number of times a character is repeated in a String
* print (String\_Name[Start:Stop:Step]) - Used for slicing operation i.e. when we want to extract only a certain portion of some text
* print (String\_Name[::-1]) - For reversing a String
* print (String\_Name.upper()) - Convert the letters of a String to upper-case
* print (String\_Name.lower()) - Convert the letters of a String to lower-case
* String Formatting in Python:- Formatting a string means to allocate the string dynamically wherever you want. The two basic approaches for formatting a string include the .format() method which accepts positional arguments for substitutions described in the curly braces and the other approach is using ‘f’ strings which is short form for format().

1. **Python Lists:-** Lists are just like arrays but in a list you can store elements of different types whereas in an array all the elements should of the same type. Lists in Python can be created by just placing the sequence inside the square brackets [ ]. Example:-

| subjects = ["English", "Science", "Hindi", "Math"]  print(subjects)  print(len(subjects)) # This will return the length of the list  print(subjects[0:2]) # This will give the values from index 0 to 2(excluding index 2) |
| --- |

Output:-

| ["English", "Science", "Hindi", "Math"]  4  ["English", "Science"] |
| --- |

* Another way to create lists is through list comprehension which is a much cleaner and efficient way of creating lists. It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. For example:-

squares = [x\*\*2 for x in range(10)] # This code will produce the square of every number in the range of 0 to 9.

1. **Python Tuple:-** Just like list, tuple is also an ordered collection of Python elements. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by the “tuple” class. All tuple operations are similar to lists, but we cannot update, delete or add an element to a Tuple. In order to create a tuple we make use of parentheses(). Eg- tuple\_1 = (1, 2, 3, 4, 5)
2. **Boolean:-** Data type with one of the two built-in values, True or False.
3. **Python Set**:- In Python, Set data type is an unordered collection of elements that is iterable, mutable and has no duplicate elements. A Set is created by placing all the items (elements) inside curly braces {}, separated by a comma. Consider the example: set\_1 = {10, “Hello”, 20, 45, “Good Morning”}. Some common set operations include:-

#### Union: Union of two sets is a set of all the elements from both sets. Union is performed using | operator.

* Intersection: Intersection of two sets is a set of elements that are common in both sets. Intersection is performed using & operator.

1. **Python Dictionary**:- Dictionary in Python is an unordered collection of data values, in the form of key-value pairs. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a ‘comma’. An empty dictionary can be created by just placing it in curly braces{}. Example:- dict\_1 = {'Name' : 'Arun Singh', 'Age' : 33, ‘Gender’ : ‘Male’}. Some common dictionary methods include:-

* clear()
* copy()
* values()
* update()
* fromkeys()
* get()
* items()
* keys()
* pop()
* popitem()

**Python Operators**

1. Arithmetic Operators- They are used for performing mathematical operations like addition, subtraction, multiplication, and division. It includes:-

* + (Addition)- adds two operands. Syntax: var\_1 + var\_2
* - (Subtraction)- subtracts two operands. Syntax: var\_1 - var\_2
* \* (Multiplication)- multiplies two operands. Syntax: var\_1 \* var\_2
* / (Division [float])- divides the first operand by the second. Syntax: var\_1 / var\_2
* // (Division [floor])- divides the first operand by the second and returns the result as a whole number. Syntax: var\_1 // var\_2
* % (Modulus)- returns the remainder when the first operand is divided by the second. Syntax: var\_1 % var\_2
* \*\* (Power)- Returns first raised to power second. Syntax: var\_1 \*\* var\_2

1. Comparison Operators- Comparison operators are responsible for comparing the values. It either returns **True** or **False** according to the condition.

* > (Greater than)- True if the left operand is greater than the right. Syntax: var\_1 > var\_2
* < (Less than)- True if the left operand is less than the right. Syntax: var\_1 < var\_2
* == (Equal to)- True if both operands are equal. Syntax: var\_1 == var\_2
* != (Not Equal to)- True if both operands are not equal. Syntax: var\_1 != var\_2
* >= (Greater than or equal to)- True if the left operand is greater than or equal to the right. Syntax: var\_1 >= var\_2
* <= (Less than or equal to)- True if the left operand is less than or equal to the right. Syntax: var\_1 <= var\_2

1. Logical Operators- These operators are used for combining conditional statements.

* and (Logical AND)- True if both the operands are true Syntax: var\_1 and var\_2
* or (Logical OR)- True if either of the operands is true. Syntax: var\_1 or var\_2
* not (Logical NOT)- True if the operand is false . Syntax: not var\_1

1. Bitwise Operators- Bitwise Operators act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers. It includes:

* & (Bitwise AND)- Syntax: var\_1 & var\_2
* | (Bitwise OR)- Syntax: var\_1 | var\_2
* ~ (Bitwise NOT)- Syntax: ~var\_1
* ^ (Bitwise XOR)- Syntax: var\_1 ^ var\_2
* >> (Bitwise right shift)- Syntax: var\_1 >>
* << (Bitwise left shift)- Syntax: var\_1 <<

1. Assignment Operators- They are used to assign values to the variables. It includes:-

* = - Syntax: var\_3 = var\_1 + var\_2
* += - Syntax: var\_1 += var\_2, var\_1 = var\_1 + var\_2
* -= - Syntax: var\_1 -= var\_2, var\_1 = var\_1 - var\_2
* \*= - Syntax: var\_1 \*= var\_2, var\_1 = var\_1 \* var\_2
* /= - Syntax: var\_1 /= var\_2, var\_1 = var\_1 / var\_2
* %= - Syntax: var\_1 %= var\_2, var\_1 = var\_1 % var\_2
* //= - Syntax: var\_1 //= var\_2, var\_1 = var\_1 // var\_2
* \*\*= - Syntax: var\_1 \*\*= var\_2, var\_1 = var\_1 \*\* var\_2

1. Identity Operators- They are used to check if two values are located on the same part of the memory. It includes:-

* **is** True if the operands are identical
* **is not** True if the operands are not identical

Example:-

| a = 10 b = 20 c = a   print(a is not b) # True print(a is c) # True |
| --- |

1. Membership Operators- They are used to test whether a value or variable is in a sequence. It includes:-

* **in** True if value is found in the sequence
* **not in** True if value is not found in the sequence

**Python OOPS Concepts**

OOPs stands for Object Oriented Programming which aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. It can also be referred to as an approach to creating neat and reusable code instead of a redundant one. This concept is also known as DRY (Don't Repeat Yourself). It includes:-

a) 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.

-> Class variable is a variable that is shared by all the different objects/instances of a class.

-> Instance variables are variables which are unique to each instance. It is defined inside a method and belongs only to the current instance of a class.

-> Methods are also called functions which are defined in a class and describe the behavior of an object.

For creating a class, consider the below example:

| class Car:   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 c1.displayInfo() |
| --- |

b) 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. Syntax: obj = className() # obj is the name of the object as in the above example.

c) Inheritance - By using inheritance, we can create a class which uses all the properties and behavior of another class. The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class. It greatly helps in the reusability of code. To inherit a class all we have to do is specify the name of the parent class in parenthesis with respect to the child class. Let's consider an example:-

| class Car: #parent class   def \_\_init\_\_(self, name, mileage):  self.name = name   self.mileage = mileage    def description(self):   return f"The {self.name} car gives the mileage of {self.mileage}km/l"  class Maruti(Car): #child class  pass  class Toyota(Car): #child class  def toyota\_desc(self):  return "This is the description method of class Toyota."   obj1 = Maruti("Maruti 800",39.53) print(obj1.description())  obj2 = Toyota("Toyota L",14) print(obj2.description()) print(obj2.audi\_desc()) |
| --- |

Inheritance comes in multiple types which are:-

-> Single level inheritance- It enables a derived class to inherit characteristics from a single parent class.

-> Multi-level inheritance- It enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from its parent class.

-> Hierarchical level inheritance- It enables more than one derived class to inherit properties from a parent class.

-> Multiple level inheritance- It enables one derived class to inherit properties from more than one base class.

d) Polymorphism- In OOP it refers to the functions having the same names but carrying out different functionalities. Suppose, we need to find the area of a shape but there are multiple shape options (rectangle, square, circle). However we could use the same method to calculate the area of any shape. This concept is called Polymorphism. Also there are two types of polymorphism

-> Compile-time Polymorphism- It is also called static polymorphism which gets resolved during the compilation time of the program. One common example is “method overloading”.

-> Run-time Polymorphism- It is also called dynamic polymorphism where it gets resolved into the run time. One common example of Run-time polymorphism is “method overriding”.

e) Encapsulation- It basically means clubbing all the necessary attributes, methods together at one single place(like class) and protecting it from the access of outsiders. Such as if an organization wants to protect an object/information from unwanted access by clients or any unauthorized person then encapsulation is the way to ensure this. In order to do we can our attributes protected by using a single underscore ( \_ ) before their names such as self.\_name or def \_method( ) or we can them private by using double underscore ( \_\_ ) in the prefix such as self.\_\_name or def \_\_method(). Example-

| class car:  def \_\_init\_\_(self, name, mileage):  self.\_name = name #protected variable  self.mileage = mileage    def description(self):   return f"The {self.\_name} car gives the mileage of {self.mileage}km/l" |
| --- |

f) Data abstraction- This concept is mainly used for hiding the internal details or implementations of a function and showing its necessary functionalities only. This is similar to the way we know how to drive a car without knowing the background mechanism or we know how to turn on or off a light using a switch but you don’t know what is happening behind the socket.

**Python Modules**

In Python, Modules are simply files with the “.py” extension containing Python code that can be imported inside another Python Program using the "import" statement. We can declare multiple functions, classes etc in a module. By using modules, the need to write the same logic again and again gets eliminated. Python provides various built-in modules for certain specific functionalities like Operating system management, disk input/output etc, generating random numbers, mathematical operations etc. Some of the most common built-in modules are:

a) OS- Python OS module provides various functions that allow us to interact and get Operating System information and even control processes up to a limit. The functions OS module provides allows us to operate on underlying Operating System tasks, irrespective of it being a Windows Platform, Macintosh or Linux.

b) MATH- The Python math module offers us the ability to perform common and useful mathematical calculations within our application like calculating combinations and permutations using factorials, trigonometric functions etc. Example-

| import math print(math.pi) |
| --- |

Output: 3.141592653589793

c) RANDOM- This is an in-built python module which is used to generate random numbers. These are pseudo-random numbers meaning these are not truly random. This module can be used to perform random actions such as generating random numbers, print random values for a list or string, etc. Example-

| import random print(random.randint(10,80)) # Code generates random integer numbers b/w 10 to 80 |
| --- |

d) DATETIME- The Python datetime module contains various classes to represent and manipulate dates and times. The key classes in datetime are date, time, datetime, timedelta, tzinfo, timezone. To use this module, we write the statement- import datetime. The datetime module is very useful for dealing with different types of dates and times format. Example:-

| from datetime import date   today = date.today()   print("Today's date is", today)  Output: Today's date is 2022-03-01 |
| --- |

**Practice Programs**

Q.1 Python program to count the number of characters (character frequency) in a string.

Sol:-

| str\_1 = "Mississippi" char\_count = {}  for i in str\_1:  if i in char\_count.keys():  char\_count[i] += 1  else:  char\_count[i] = 1  print(char\_count) |
| --- |

Output:

| {'M': 1, 'i': 4, 's': 4, 'p': 2} |
| --- |

Q.2 Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.

Sol:-

| str\_1 = input("Enter the first string: ") str\_2 = input("Enter the second string: ")  new\_str\_1 = str\_2[:2] + str\_1[2:] new\_str\_2 = str\_1[:2] + str\_2[2:]  print("Resultant String is", new\_str\_1, new\_str\_2) |
| --- |

Output:-

| Enter the first string: Hello Enter the second string: World Resultant String is Wollo Herld |
| --- |

Q.3 Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings

Sol:-

| list\_items = ['1221', 'banana', 'ababa', 'xyzqa'] count = 0 for item in list\_items:  if len(item) >= 2 and (item[0] == item[-1]):  count += 1  print("No of elements having length 2 or more:", count) |
| --- |

Output:-

| No of elements having length 2 or more: 2 |
| --- |

Q.4 Python program to remove duplicates from a list

Sol:-

| list\_items = [10, 20, 30, 20, 10, 50, 60, 40, 80, 50, 40] unique\_items = list(set(list\_items)) # set function removes any duplicate elements print(unique\_items) |
| --- |

Output:-

| [40, 10, 80, 50, 20, 60, 30] |
| --- |

Q.5 Python program to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x\*x)

Sol:-

| num\_val = int(input("Enter a number: ")) dict\_values = {}  for val in range(1, num\_val+1):  dict\_values[val] = val \*\* 2  print("Values of new dictionary are:", dict\_values) |
| --- |

Output:

| Enter a number: 4 Values of new dictionary are: {1: 1, 2: 4, 3: 9, 4: 16} |
| --- |

Q.6 Python program to combine two dictionary adding values for common keys

Sol:-

| dict\_1 = {'x': 10, 'y': 25, 'z': 12, 'u': 18} dict\_2 = {'x': 30, 'y': 17, 'z': 51, 't': 8} dict\_3 = {}  dict\_1\_keys = dict\_1.keys() dict\_2\_keys = dict\_2.keys()  for key\_1 in dict\_1\_keys:  for key\_2 in dict\_2\_keys:  if key\_1 == key\_2:  dict\_3[key\_1] = dict\_1[key\_1] + dict\_2[key\_2]   print(dict\_3) |
| --- |

Output:-

| {'x': 40, 'y': 42, 'z': 63} |
| --- |

Q.7 Python program to generate a random value between two integers (inclusive) and a random multiple of 7 between 0 and 70.

Sol:-

| import random  print("Random value between two integers:") print(random.randint(10, 85)) print(random.randint(-12, 24)) print("Random multiple of 7 between 0 and 70:") print(random.randint(0, 10) \* 7) |
| --- |

Output:-

| Random value between two integers: 36 -3 Random multiple of 7 between 0 and 70: 49 |
| --- |