**Day-07**

**26-02-2025**

**================**

**Python Datatypes**

**============**

What is Datatype?

============

-> Describe the type of the data to be processed in the program to accomplish task.

Ex: Myntra application

added product into cart

 total number of products ==> 5

 total amount of cart products ==> 3997.67

 Product title ==> "Oneplus"

 etc.

-> In python, there are two categories of datatypes:

 1) Primitive Datatypes

 2) Non-Primitive Datatypes/Reference Based Datatypes

1) Primitive Datatypes

--------------------------

-> All basic or fundamental datatypes are called as "Primitive datatypes".

-> There are different formats of primitive datatypes:

 1) Number format/Numerical format

 2) Boolean Format

 3) Text Format

-> Number Format describe the data in the format of numbers (with decimal point or without decimal point).

-> Number formatted datatypes are:

 1) Integer --> pre-defined class 'int'

 2) Float --> pre-defined class 'float'

 3) Complex --> pre-defined class 'complex'

-> Integer datatype can allow to define the data without decimal point.

Ex: 1234, 1, -123, etc.

-> Float datatype can allow to define the data with decimal point.

Ex: 123.234, 0.0012, -1.23 etc.,

123.23400010101 etc.

-> Complex data is the combination of real data and imaginary data.

Syntax:

 real-data +/- imaginary-data

Note:

-----

 Imaginary number/data can always suffix with 'j'

Ex: 123 - 101j

Here: 123 ==> real data

 101j ==> imaginary data

1.001 + 1.2301j

Here:

 1.001 ==> real data

 1.2301j ==> imaginary data

100 + 1.001j

Here:

 100 ==> real data

 1.001j ==> imaginary data

-> Boolean format is also called as "Non-numerical format".

-> For this format, we just have only one type of datatype:

 boolean ----> pre-defined class "bool"

-> boolean data can allow to define with two values

 1) True

 2) False

Note:

-----

True and False ==> Keywords

a = False

print(a)

print(type(a))

-> Internally, PVM can understand:

 True ==> 1

 False ==> 0

print(True + True) # 2

print(True - True) # 0

print(True + False) # 1

print(True - False) # 1

print(False - True) # -1

print(True \* True) # 1

-> Text Formatted data in python is:

 String --> pre-defined class 'str'

-> String is the collection/group of characters

which must be enclosed with single quotes ('') or

double quotes ("") or triple quotes (''' '''/""" """).

Ex: 'a', 'abc' ==> strings

"a", "abcd" ==> strings

'''a''', '''abcde''' ==> strings

"""a""", """abcdef""" ==> strings

a = 'a'

b = 'abcd'

c = "a"

d = "123"

e = '''@#as123'''

f = """ravi@123$#"""

print(type(a), type(b), type(c))

print(type(d), type(e), type(f))

Note:

=====

In other languages character data notation is there.

But python consider the character notation as string only.

-> Python is "Dynamically Typed Programming Language"

Explanation:

------------

 High-level programming languages

 based on the data definition, these are classified into three types:

 1) Statically Typed

 2) Dynamically Typed

 3) Strongly Typed

 -> When a programming language can convert the assigned value based on the definition or type of the data, such type of programming languages are called as "Statically typed programming languages".

#include<stdio.h>

 void main(){

 int a = 1.001;

 printf("%d",a);

 }

 Ex: C

 -> When programming languages the data which we want to assign in the program must be satisfy the type what we have specified or its equivalent to the specified type before the data name. Such programming languages are called "Strongly typed programming languages".

 Ex: Java

 class TestProgram{

 public static void main(String[] args){

 int a = 1.001;

 System.out.println(a);

 }

 }

type():

-------

-> a pre-defined function

-> to know the type of the data, we can use type()

Syntax:

 type(data/data-name)

 -> For dynamically typed programming languages, while defining the data no need to be specify the type of the data explicitly before the data name.

 -> These languages can detect the type based on the assigned value.

a = 1.23

print(a)

print(type(a))

-> Every data definition is considered as an object by the PVM (Python Virtual Machine) because that every datatype have a pre-defined class in the python library.

What is an Object?

------------------

-> Object is the physical entity

-> Reference for the class.

What is the class?

------------------

-> Logical Entity

-> Blueprint

-> Reference datatype

 collection of data and methods