CH-3 Data Handling

Day- 1

Data Types

Mutable and Immutable Type

Class Work

Data Types

Numbers : The Number data types are used to store numeric values in Python. The Numbers in Python have following core data types:

1) Integer
2) Floating point
3) Complex Numbers

Integers :- Integers are whole numbers such as 5,39 etc. They have no fractional parts. Integers are represented in Python by numeric values with no decimal point. It can be positive or negative.

Floating point Numbers :- A number having fractional part is a floating – point number. It can be positive and negative. For ex 3.14

1) Fractional Form : 3500.67
2) Exponential Form:- 3.50075E03, 1.479E02

Complex Numbers :- A complex number is in the form A + Bi where i is the imaginary number, equal to square root of -1 i.e.$\sqrt{-1}$.

Complex Numbers in Python

Python represents complex numbers in the form A + Bj .

For ex:

```
A = 0 + 3.1j
B = 1.5 + 2.0j
print(A +B)
```

```
1.5 +5.1j
```

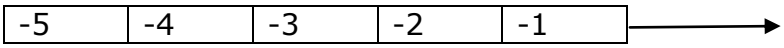
String

Any thing enclosed in ` or " quotes become a string. A string can hold any type of known characters i.e. letters, numbers and special characters.

For example :- "abcd" , "1234" ,"@@d"

0 1 2 3 4 ⟶ Forward Indexing

H	E	L	L	0



len() :- len function return the total number of char e string.

Mutable :- The mutable types are those whose values can be changed in place.

Mutable types are list, dictionary.

Immutable :- The immutable types are those that can never change their value in place. For ex : integers, floating point numbers, Boolean , strings, tuples etc.

Day 2

Variable Internals:-

1) Type :- The type of an object determines the data type of value.
   A=10
   print(type(A))
   <class int>
2) Id :- The id return the memory location of an object.
   A=10
   print(id(A))
   30899132

Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators

Arithmetic operators are used to perform arithmetic operations between two operands. It includes +(addition), -(subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**).

Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator

Comparison operators are used to comparing the value of the two operands and returns boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Python assignment operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the reminder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

DAY -3

Bitwise operator

The bitwise operators perform bit by bit operation on the values of the two operands.

For example,

```
if a = 7;
   b = 6;
then, binary (a) = 0111
     binary (b) = 0011

hence, a & b = 0011
       a | b = 0111
            a ^ b = 0100
```

9. ~ a = 1000

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero otherwise the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different otherwise the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

Logical Operators

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true then not (a) will be false and vice versa.

DAY -4 AND 5

Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or

	dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both side do not point to the same object.

Operator Precedence

The precedence of the operators is important to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in python is given below.

Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus and minus
>> <<	Left shift and right shift
&	Binary and.
^ |	Binary xor and or
<= < > >=	Comparison operators (less then, less then equal to, greater then, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators

not or and	Logical operators

DAY 6

CH-4

IF – CONDITION :- IF Check the condition is true or false. If the condition is true the statement inside the if block get execute and if it is false nothing is display in the screen.

Syntax:

```
if(condition)
 {
Statement
}
```

Ex :-

```
A=int(input("Enter A:"))
B=int(input("Enter B:"))
if(A > B)
{
print("A is greater than B")
}
```

Day- 2

- Barebones of a Python Program
    - Expressions
    - Statements
    - Comments
    - Function
    - Blocks and indentation
- Variables and Assignment

Expression: - An expression is any legal combination of symbols that represents a value.

Example :- 15, 2.9

Statements :- A statement is a programming instruction that does some thing i.e. some action takes place.

Example :- print("hello")

Comments :- Comments are the additional readable information to clarify the source code.

Comment in Python begin with symbol # and generally end with end of physical line.

In the above code, you can see four comments :

(I) The physical lines beginning with # are the full line comments. There are three full line comments in the above program are :

```
# This program shows a program' s components

#Definition of function SeeYou( ) follows

#Main program code follows now
```

(II) The fourth comment is an inline comment as it starts in the middle of a physical line, after Python code (see below)

```
If  b < 5: # colon means it requires a block
```

Multiline comments :- Multiline comment in two ways :-

i) Add a # symbol in the beginning of every physical line part of the multi-line comments e.g.

```
# Multi-line comments are useful for detailed additional information.
# Related to the program in question.
#I thelps clarify certain important things
```

ii) Type comment as a triple quoted multi-line string e.g.

```
``` Multi –line comments are useful for detailed additional information related to the program in question. It helps clarify certain important things
```
```

Functions :- A function is a code that has a name and it can be reused by specifying its name in the program, where needed.

For ex Seeyou() # function-call statement

Block and Indentation :- Sometimes a group of statements is part of another statement or function. Such a group of one or more statements is called block or code-block or suite.

For ex :-

```
if  b < 5:
    print (“Value of b is less than 5”)
    print(“Thank you”)
```

Variables and Assignment :-

Python Variables

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

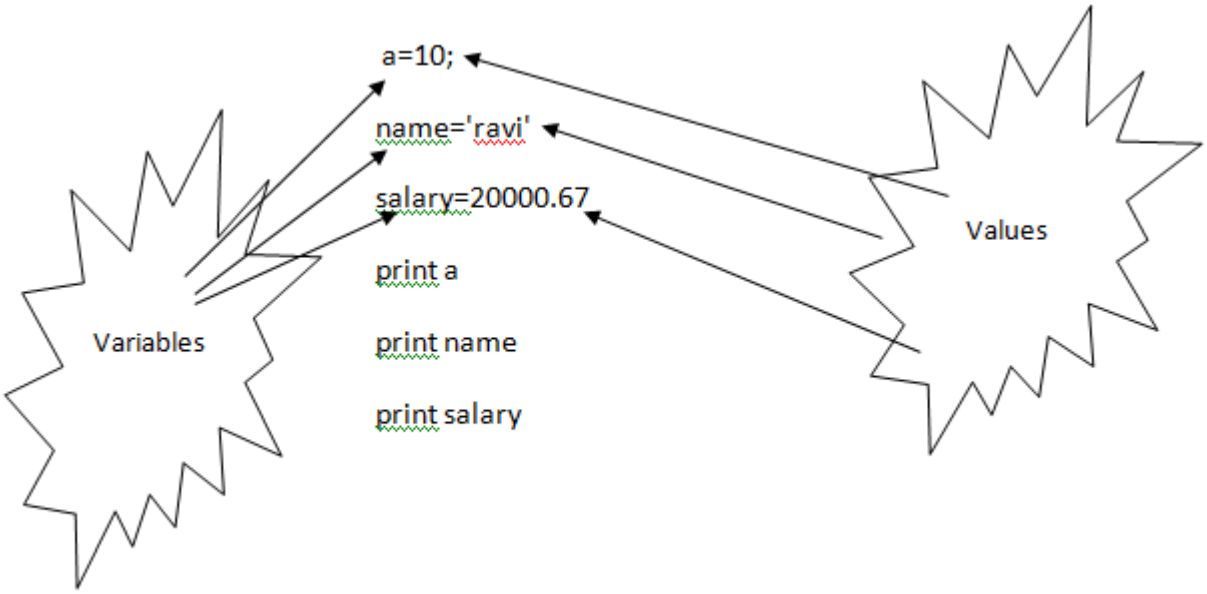
Declaring Variable and Assigning Values

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Eg:



Output:

```
>>>
10
ravi
20000.67
>>>
```

Activity :-

Program to assign a value to variable and print its value.

Assignment:

Do the worksheet given in the class.

Day- 3

- Multiple assignment
- Variable definition
- Dynamic Typing
- Simple Input and output

Class Work

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples.

1. Assigning single value to multiple variables

Eg:

```
1. x=y=z=50
2. print iple
3. print y
4. print z
```

Output:

```
1. >>>
2. 50
3. 50
4. 50
5. >>>
```

2.Assigning multiple values to multiple variables:

Eg:

```
1. a,b,c=5,10,15
2. print a
3. print b
4. print c
```

Output:

```
1. >>>
2. 5
3. 10
4. 15
5. >>>
```

Variable definition :- A variable is defined only when you assign some value to it. Using an undefined variable in an expression /statement cause an error called NameError.

```
X=20
print(X)
```

Dynamic Typing: - Python is a **dynamically typed** language. It doesn't know about the type of the variable until the code is run. So declaration is of no use. What it does is, It stores that value at some memory location and then binds that variable name to that memory container. And makes the contents of the container accessible through that variable name. So the data type does not matter. As it will get to know the type of the value at run-time.

```
        # This will store 6 in the memory and binds the

# name x to it. After it runs, type of x will
# be int.
x = 6

print(type(x))

# This will store 'hello' at some location int
# the memory and binds name x to it. After it
# runs type of x will be str.
x = 'hello'

print(type(x))
```

Output:

```
<class 'int'>
<class 'str'>
```

Simple Input and output:- Python provides numerous built-in functions that are readily available to us at the Python prompt.

Some of the functions like `input()` and `print()` are widely used for standard input and output operations respectively. Let us see the output section first.

Python Output Using print() function

We use the `print()` function to output data to the standard output device (screen).
We can also output data to a file, but this will be discussed later. An example use is given below.

print('This sentence is output to the screen')

 # Output: This sentence is output to the screen

a = 5

print('The value of a is', a)

 # Output: The value of a is 5

Python Input

Up till now, our programs were static. The value of variables were defined or hard coded into the source code.

To allow flexibility we might want to take the input from the user. In Python, we have the `input()` function to allow this. The syntax for `input()` is

```
input([prompt])
```

where `prompt` is the string we wish to display on the screen. It is optional.

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
'10'
```

Here, we can see that the entered value `10` is a string, not a number. To convert this into a number we can use `int()` or `float()` functions.

```
>>> int('10')
10
>>> float('10')
```

```
10.0
```

This same operation can be performed using the `eval()` function. But it takes it further. It can evaluate even expressions, provided the input is a string

```
>>> int('2+3')
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '2+3'
>>> eval('2+3')
5
```