Session 5 & 6: Python Programming

Note:

Escape sequence:
Incase of strings, when we need to represent special characters we use escape sequences, denoted by backslash '\'.
Example:
```
a= 'Python\'s classes'
print(a)
a= "Python\nclass"
print(a)
```

Output:
```
Python's classes
Python
class
```

Here, such string literals are called escape sequences.
In python, we have multiple escape sequences such as:

\n : newline
\t : tab
\r : carriage return
\b : backspace
\v : vertical tab
\' : single quote
\" : double quote
\\ : slash

Example:
```
a = 'Python\\Lava class'
print(a)
```

Output:
```
Python\Lava class
```

Constants:
In python, there isn't actually any provision of constants. But to make working easier for the programmer, they usually denote constants in all capital letters.

Example:

```
MAX_VALUE = 10
```

Comments:
There are single line comments(#) and multi-line comments (ctrl (or command) + backslash).

Operators

Operators are those symbols that carry out certain operations.
Python supports a wide range of operators as follows:

• Arithmetic Operators
• Comparison /Relational Operators
• Logical Operators
• Assignment Operators
• Bitwise Operators
• Shift Operators
• Equality Operators
• Ternary Operator
• Special Operators
  - Membership Operators
  - Identity Operators

1. **Arithmetic operators:**
   In Python, arithmetic operators are used to perform basic mathematical operations.

   + : Addition
   - : Subtraction
   * : Multiplication
   / : Division (always gives value in float)
   // : Floor Division (if atleast one value is float then the result is float as well, else it is int)
   % : Modulus
   ** : Exponentiation

   Example:
   ```
   print(12 + 6)
   print(12 - 6)
   print(12 * 6)
   ```

```
print(12 / 6)
print(12 // 6)
print(12 % 6)
print(12 ** 6)
```

```
Output:
   18
   6
   72
   2.0
   2
   0
   2985984
```

Incase of string:

```
str1='Ram '
str2='Sita'
print(str1+str2)
print(str1 * 4) #to multiply, there needs to be one string and one int
(only int)
print(str1 + '10')
print(str1 * int('10'))
print(str1 * True)
print(str1 * False)   #denotes 0 times repetition
```

```
Output:
   Ram Sita
   Ram Ram Ram Ram
   Ram 10
   Ram Ram Ram Ram Ram Ram Ram Ram Ram Ram
   Ram
```

2. **Comparative/ Relational operators:**
   Comparison operators are used to compare two values. The result of the comparison is a Boolean value i.e. True or False. It includes:

   > : Greater than
   < : Less than
   >= : Greater than or equal to
   <= : Less than

Example 1:

```
            Output
a= 10
b= 20
print(a>b)       //False
print(a>=b)      //True
```

Example 2: (Strings can also be compared, they use ASCII code for comparison but string cannot be compared with integers)

```
            Output
a = “Hello”
b= “world”
print(a>b)       //False (compares the first letter)
print(a<=b)      //True (compares the first letter)
```

Example 3:

```
            Output
a= True
b= True
print(a<=b)      //True
print(a<b)       //False
```

Chaining of relational operation:

In case of chaining of relational operation, the output is True if and only if all the comparisons are True.

Example:

```
                            Output
print(10<20<30<40)                //True
print(10<20<30<40>50<60<70) //False
```

3. **Equality Operator:**
   It includes:
   == : Equal to
   != : Not equal to

   Example:

```
                            Output
```

```
print(10==20)              //False
print(10!=20)              //True
print(10==True)            //False
print(1==True)             //True
print(10 == 10.0)          //True
print(False==False)        //True
print(10==10!=20!=10)      //True
```

4. **Logical Operator:**
   It includes: and, or, not
   We use logical operators for boolean and non-boolean types.

   Boolean Types Example:

```
                           Output
print(True and True)       //True
print(True and False)      //False
print(True or True)        //True
print(True or False)       //True
print(not False)           //True
```

   Non Boolean Types:
   (Note: 0-False, non 0: True)

   - x and y
     If x is false then return x,
     If x is true then return y.

     Example:

```
                           Output
print(10 and 20)           //20
print(0 and 20)            //0
print(‘ ’ and 20)          //     (empty string)
print([ ] and 20)          //[ ]
print(‘hello’ and ‘python’)//python
```

   - x or y
     If x is true then result is x,
     If x is false then result is y.

Example:

```
                           Output
print(10 or 20)            //10
print(0 or 20)             //20
print(‘ ’ or 20)           //20
print([ ] and 20)          //20
print([10 ] and 20)        //[10]
print(‘hello’ or ‘python’) //hello
print(‘hello’ or ‘ ’)      //hello
```

- not
  If x is false, not x gives True
  If x is true, not x gives False

  Example:

```
                      Output
print(not 10)         //False
print(not ‘ ’)        //True
print(not 0)          //True
print(not [10,20])    //False
```

5. **Bitwise Operator:**
   This operator works in bits. Bitwise operators only work in int or bool data types.
   It includes:

& : Bitwise and	(if both bits are 1 then result is 1, else 0)
| : Bitwise or	(if atleast one bit is 1 then result is 1, else 0)
^ : Bitwise x-or	(if bits are different then result is 1, else 0)
~ : Bitwise not/complement	(if bit is 1 then result is 0, and vice versa)

Example:

```
                Output
print(4&5)      //4
print(4|5)      //5
print(4^5)      //1
print(~4)       //-5
print(~-4)      //3
```

6. **Shift Operator:**
   << : left shift operator
   >> : right shift operator

   Example:

```
                      Output:
print(10<<2)               //40
print(10>>2)               //2
```

7. **Assignment Operator:**
   = : assigning a value
   Example:

```
                Output
a=10
print(a)        //10
```

   Compound assignment operator:
   If arithmetic operator or bitwise operator comes with the assignment operator, then it is known as compound assignment operator.
   Example:

```
                Output
a=10
a+=10
print(a)        //20

x=10
x**=2
print(x)        //100

y=10
y&=5
print(y)        //0
```

   Increment and decrement is not supported in python: x++, x - -
   But , in the case of ++++x, it is supported by python, not as increment or decrement but as sign.

   Example:

```
                Output
a=10
```

```
print(--a)      //10
print(---a)     //-10
```

8. **Ternary Operator/ Conditional Operator:**
   The operator with three operands, then it is known as ternary operator. As ternary operators check conditions, it is also known as conditional operator.
   Syntax:
   ```
   x= first_value if condition else second_value
   ```
   Here, operands are first_value, condition and second_value

   Example 1:
   ```
   a=10
   b=20
   c=10 if a<b else 20
   print(c)
   ```
   Output:
   ```
   10
   ```

   Example 2:
   ```
   a=int(input("Enter the first number: "))
   b=int(input("Enter the second number: "))
   min = a if a<b else b
   print(min)
   ```
   Output:
   ```
   Enter the first number: 30
   Enter the second number: 20
   20
   ```

   Nesting of ternary operation:

   Example 1:
   ```
   a=int(input("Enter the first number: "))
   b=int(input("Enter the second number: "))
   c=int(input("Enter the third number: "))
   min = a if a<b and a<c else b if b<c else c
   print("The minimum value is : ", min)
   ```

   Output:
   ```
   Enter the first number: 10
   Enter the second number: 20
   ```

```
Enter the third number: 40
The minimum value is :  10
```

This process, however, cannot be done as:

```
a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))
c=int(input("Enter the third number: "))
min = a if a<b<c else b if b<c else c
print("The minimum value is : ", min)
```

Output:

```
Enter the first number: 5
Enter the second number: 35
Enter the third number: 30
The minimum value is :  30
```

Here, the minimum value only depends on the comparison between b and c, which should not be the case.

Example 2:

```
a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))
print("Both numbers are equal" if a==b else "First number is
less" if a<b else "First number is greater")
```

Output:

```
Enter the first number: 40
Enter the second number: 39
First number is greater
```

9. **Special operators:**

- **Identity Operator:**
  It is used to compare addresses and includes: is, is not.

  Example 1:

  ```
  a=10
  b=10
  print(id(a))
  ```

```
print(id(b))
print(a is b)
print(a is not b)
```
Output:
```
True
```

Example 2:
```
a=[10,20,30]
b=[10,20,30]
print(id(a))
print(id(b))
print(a is b)
print(a is not b)
```
Output:
```
4378020160
4378021952
False
True
```

- **Membership Operator:**
  It includes: in , not in

  Example:
```
x= 'Python is fun'
print('s' in x)
print('fun' in x)
print('Python' not in x)
a=["dipesh","isha","kabin"]
print('isha' in a)
```
  Output:
```
True
True
False
True
```

Operator Precedence in Python:

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

When two operators have the same precedence, associativity helps to determine the order of operations. Almost all the operators have left-to-right associativity.

Note:
Module is a collection of functions, variables, class, etc. We use modules for reusability.
We can create our own modules or use a pre-built module in python.
Math Module is one of such modules.

Example 1:

```
import math
r=2
print("Area of circle is:", math.pi * math.pow(r,2))
print(dir(math))    //shows all directories under the math module
```

Output:

```
Area of circle is:  12.566370614359172
['__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'cbrt', 'ceil', 'comb', 'copysign', 'cos', 'cosh',
'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1',
'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan',
'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod',
'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'sumprod', 'tan',
'tanh', 'tau', 'trunc', 'ulp']
```

Example 2 (we can import in other way as well, along with aliasing):

```python
from math import sqrt as s
print(s(16))
```

Output:

```
4.0
```