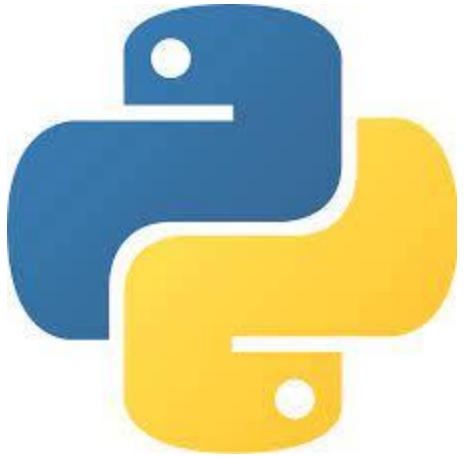

Python Programming

LECTURE-7

19TH OCT 2021

KIRTI SHARMA

KRTBHARDWAJ1@GMAIL.COM

Topics Covered

- Operators
- Expressions

Python 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 specific programming language.

Python provides a variety of operators, which are 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 (**) 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 results 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 = 2.0
* (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 the 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.

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.

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand.

Operator	Description
=	It assigns 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 assigns 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 assigns 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 assigns the modified value back to then the 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 assigns the reminder back to the 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.

Bitwise Operators

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) = 0110

hence, a & b = 0011
a | b = 0111
a ^ b = 0100
~ 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.

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.

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

The identity operators are used to decide whether an element certain class or type.

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 sides do not point to the same object.

Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first.

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 than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

InPut and Output

Sometimes, a program needs to interact with the user's to get some input data or information from the end user and process it to give the desired output. In Python, we have the input() function for taking the user input. The input() function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the input() function treats them as strings only.

The syntax for input() is:

input ([Prompt])

Prompt is the string we may like to display on the screen prior to taking the input, and it is optional. When a prompt is specified, first it is displayed on the screen after which the user can enter data. The input() takes exactly what is typed from the keyboard, converts it into a string and assigns it to the variable on left-hand side of the assignment operator (=). Entering data for the input function is terminated by pressing the enter key.

```
>>> fname = input("Enter your first name: ")

Enter your first name: Arnab

>>> age = input("Enter your age: ")

Enter your age: 19

>>> type(age)

<class 'str'>
```

The variable fname will get the string 'Arnab', entered by the user. Similarly, the variable age will get the string '19'. We can typecast or change the datatype of the string data accepted from user to an appropriate numeric value. For example, the following statement will convert the accepted string to an integer. If the user enters any non-numeric value, an error will be generated.

Python uses the print() function to output data
to
standard output device — the screen.

The function print() evaluates the expression before displaying it on the screen. The print() outputs a complete line and then moves to the next line for subsequent output. The syntax for print() is:

print(value [, ..., sep = ' ', end = '\n'])

- sep: The optional parameter sep is a separator between the output values. We can use a character, integer or a string as a separator. The default separator is space.
- end: This is also optional and it allows us to specify any string to be appended after the last value. The default is a new line.

Statement	Output
`print("Hello")`	`Hello`
`print(10*2.5)`	`25.0`
`print("I" + "love" + "my" + "country")`	`Ilovemycountry`
`print("I'm", 16, "years old")`	`I'm 16 years old`

Type Conversion

Consider the following program

```
num1 = input("Enter a number and I'll double it: ")
num1 = num1 * 2
print(num1)
```

The program was expected to display double the value of the number received and store in variable num1. So if a user enters 2 and expects the program to display 4 as the output, the program displays the following result:

```
Enter a number and I'll double it: 2
```

This is because the value returned by the input function is a string ("2") by default. As a result, in statement num1 = num1 * 2, num1 has string value and * acts as repetition operator which results in output as "22". To get 4 as output, we need to convert the data type of the value entered by the user to integer.

Thus,
we modify the program as follows:

```
num1 = input("Enter a number and I'll double it: ")
```

num1 = int(num1)	#convert string input to #integer

```
num1 = num1 * 2
print(num1)
```

Now, the program will display the expected output as follows:

```
Enter a number and I'll double it: 2
4
```