Python Programming

LECTURE-8

21ST OCT 2021

KIRTI SHARMA

KRTBHARDWAJ1@GMAIL.COM

Topics Covered

- ❑ Expressions
- ❑ Operator Precedence
- ❑ Python if-else statements

eXPressIons

An expression is defined as a combination of constants, variables, and operators. An expression always evaluates to a value. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression. Some examples of valid expressions are given below.

(i) 100 (ii) num (iii) num – 20.4	(iv) 3.0 + 3.14 (v) 23/3 -5 * 7(14 -2) (vi) "Global" + "Citizen"

Operator Precedence

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

Evaluation of the expression is based on precedence of operators. When an expression contains different kinds of operators, precedence determines which operator should be applied first. Higher precedence operator is evaluated before the lower precedence operator.

Most of the operators studied till now are binary operators. Binary operators are operators with two operands.

The unary operators need only one operand, and they have a higher precedence than the binary operators. The minus (-) as well as + (plus) operators can act as both unary and binary operators, but not is a unary logical operator.

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

Note:

a) Parenthesis can be used to override the precedence of operators. The expression within () is evaluated first.

b) For operators with equal precedence, the expression is evaluated from left to right.

How will Python evaluate the following expression?

20 + 30 * 40

How will Python evaluate the following expression?

20 + 30 * 40

Solution:

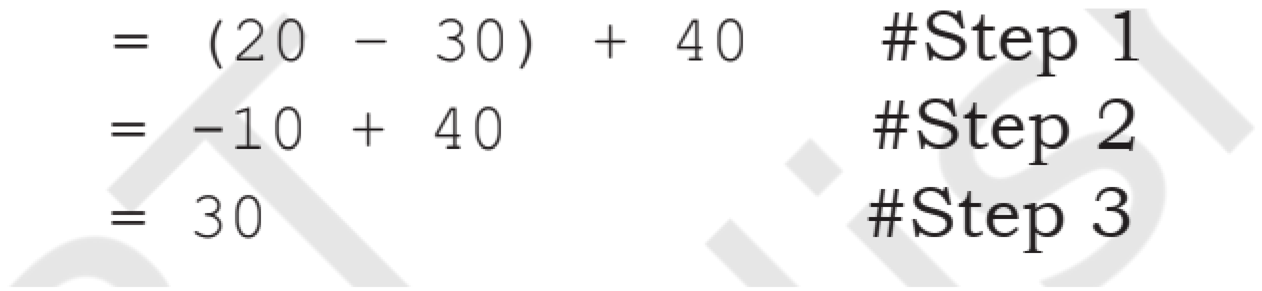
```
= 20 + (30 * 40)          #Step 1
#precedence of * is more than that of +
= 20 + 1200               #Step 2
= 1220                    #Step 3
```

How will Python evaluate the following expression?

20 - 30 + 40

Solution:

The two operators (–) and (+) have equal precedence. Thus, the first operator, i.e., subtraction is applied before the second operator, i.e., addition (left to right).



```
= (20 – 30) + 40     #Step 1
= -10 + 40           #Step 2
= 30                 #Step 3
```

Statement

In Python, a statement is a unit of code that the Python interpreter can execute.

```
>>> x = 4 #assignment statement

>>> cube = x ** 3 #assignment statement

>>> print (x, cube) #print statement

4 64
```

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.

print() function

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
22
```

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
```

Let us now understand what is type conversion and how it works. As and when required, we can change the data type of a variable in Python from one type to another.

Such data type conversion can happen in two ways:

either explicitly (forced) when the programmer specifies for the interpreter to convert a data type to another type; or implicitly, when the interpreter understands such a need by itself and does the type conversion automatically.

Explicit Conversion

Explicit conversion, also called type casting happens when data type conversion takes place because the programmer forced it in the program. The general form of an explicit data type conversion is:

```
(new_data_type) (expression)
```

With explicit type conversion, there is a risk of loss of information since we are forcing an expression to be of a specific type. For example, converting a floating value of x = 20.67 into an integer type, i.e., int(x) will discard the fractional part .67.