4. Python – Basic Operators

Python language supports following type of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Python Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Python Comparison Operators:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Python Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	c = a + b will assigne value of a + b into c
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//=	Floor Division and assigns a value, Performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(a | b) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(a ^ b) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~a) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	a << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 will give 15 which is 0000 1111

Python Logical Operators:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

Python Membership Operators:

In addition to the operators discussed previously, Python has membership operators, which test for membership in a sequence, such as strings, lists, or tuples.

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here **in** results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here **not in** results in a 1 if x is a member of sequence y.

Python Operators Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Python – IF...ELIF...ELSE Statement

- The syntax of the if statement is:

```
if expression:
    statement(s)
```

```
if expression:
    statement(s)
else:
    statement(s)
```

```
Example:
   var1 = 100
   if var1:
       print "1 - Got a true expression value"
       print var1
   var2 = 0
   if var2:
       print "2 - Got a true expression value"
       print var2
print "Good bye!"
```

```
var1 = 100
if var1:
   print "1 - Got a true expression value"
   print var1
else:
   print "1 - Got a false expression value"
   print var1

var2 = 0
if var2:
   print "2 - Got a true expression value"
   print var2
else:
   print "2 - Got a false expression value"
   print var2
print "Good bye!"
```

The Nested *if...elif...else* Construct

```
Example:
var = 100
if var < 200:
   print "Expression value is less than 200"
   if var == 150:
      print "Which is 150"
   elif var == 100:
      print "Which is 100"
   elif var == 50:
      print "Which is 50"
elif var < 50:
   print "Expression value is less than 50"
else:
   print "Could not find true expression"

print "Good bye!"
```

Single Statement Suites:

- If the suite of an **if** clause consists only of a single line, it may go on the same line as the header statement:

```
if ( expression == 1 ) : print "Value of expression is 1"
```

5. Python – while Loop Statements

- The **while** loop is one of the looping constructs available in Python. The **while** loop continues until the expression becomes false. The expression has to be a logical expression and must return either a *true* or a *false* value

  The syntax of the while loop is:

```
while expression:
    statement(s)
```

Example:

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1
print "Good bye!"
```

The Infinite Loops:

- You must use caution when using while loops because of the possibility that this condition never resolves to a false value. This results in a loop that never ends. Such a loop is called an infinite loop.
- An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Following loop will continue till you enter CTRL+C :

```
while var == 1 :  # This constructs an infinite loop
   num = raw_input("Enter a number  :")
   print "You entered: ", num
print "Good bye!"
```

Single Statement Suites:

- Similar to the **if** statement syntax, if your **while** clause consists only of a single statement, it may be placed on the same line as the while header.

- Here is the syntax of a one-line while clause:

```
while expression : statement
```

6. Python – for Loop Statements

- The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- The syntax of the loop look is:

```
for iterating_var in sequence:
    statements(s)
```

Example:

```
for letter in 'Python':     # First Example
   print 'Current Letter :', letter

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
   print 'Current fruit :', fruit
print "Good bye!"
```

Iterating by Sequence Index:

- An alternative way of iterating through each item is by index offset into the sequence itself:

- **Example:**

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
   print 'Current fruit :', fruits[index]

print "Good bye!"
```

7. Python *break,continue and pass* Statements

The *break* Statement:

- The **break** statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

Example:

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print 'Current Letter :', letter
var = 10                    # Second Example
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break
print "Good bye!"
```

The *continue* Statement:

- The **continue** statement in Python returns the control to the beginning of the while loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Example:

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print 'Current Letter :', letter
var = 10                    # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print 'Current variable value :', var
print "Good bye!"
```

The *else* Statement Used with Loops

- Python supports to have an **else** statement associated with a loop statements.
  - If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
  - If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

Example:

```
for num in range(10,20):  #to iterate between 10 to 20
   for i in range(2,num): #to iterate on the factors of the number
      if num%i == 0:      #to determine the first factor
         j=num/i #to calculate the second factor
         print '%d equals %d * %d' % (num,i,j)
         break #to move to the next number, the #first FOR
   else:         # else part of the loop
      print num, 'is a prime number'
```

The *pass* Statement:

- The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

Example:

```
for letter in 'Python':
   if letter == 'h':
      pass
      print 'This is pass block'
   print 'Current Letter :', letter
print "Good bye!"
```