



Data Analysis With Python

© Confidentiality & Proprietary Information

This document contains information that is Proprietary and confidential ("Confidential Information") to Boston Training Academy and shall not be used or disclosed outside.

Further, the Confidential Information should not be transmitted, duplicated, or used in whole or in part for any purpose other than what it is intended for herein. Any use or disclosure in whole or in part of this Confidential Information without the express written permission of Boston Training Academy is strictly prohibited.

This is a confidential document prepared by Boston Training Academy.

The illustrative formats and examples have been created solely to simulate Learning and do not purport to represent/reflect on work practices of any particular party/parties.

Unauthorized possession of the material or disclosure of the Proprietary information may result in legal action.

© Boston Training Academy 2021

Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression $4 + 5 = 9$. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a look on all operators one by one.

Python Arithmetic Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then –

[[Show Example](#)]

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to !=

		operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	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	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply	It multiplies right operand with the left operand and assign	c *= a is equivalent

AND	the result to left operand	to $c = c * a$
/= Divide AND	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	It takes modulus using two operands and assign the result to left operand	$c \% = a$ is equivalent to $c = c \% a$
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	$c ** = a$ is equivalent to $c = c ** a$
//= Floor Division	It performs floor division on operators and assign value to the left operand	$c //= a$ is equivalent to $c = c // a$

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if $a = 60$; and $b = 13$; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands –

$a = 0011\ 1100$

$b = 0000\ 1101$

$a \& b = 0000\ 1100$

$a | b = 0011\ 1101$

$a \wedge b = 0011\ 0001$

$\sim a = 1100\ 0011$

There are following Bitwise operators supported by Python language

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
Binary OR	It copies a bit if it exists in either operand.	(a b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left	The left operands value is moved left by the number of	$a \ll 2 = 240$ (means

Shift	bits specified by the right operand.	1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

[[Show Example](#)]

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

[[Show Example](#)]

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 not a member of sequence y.

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below –

[[Show Example](#)]

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

[[Show Example](#)]

Sr.No.	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Complement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % // Multiply, divide, modulo and floor division
4	+ - Addition and subtraction
5	>> << Right and left bitwise shift
6	& Bitwise 'AND'
7	^ Bitwise exclusive 'OR' and regular 'OR'
8	<= < > >= Comparison operators
9	<> == != Equality operators

10	= %= /= //= -= += *= **= Assignment operators
11	is is not Identity operators
12	in not in Membership operators
13	not or and Logical operators