# **Python Operators**

Learning the operators is an excellent place to start to learn Python. Operators are special symbols that perform specific operations on one or more operands (values) and then return a result. For example, you can calculate the sum of two numbers using an addition (+) operator.

Python has seven types of operators that we can use to perform different operation and produce a result.

1. Arithmetic operator
2. Relational operators
3. Assignment operators
4. Logical operators
5. Membership operators
6. Identity operators
7. Bitwise operators

## Arithmetic operator

Arithmetic operators are the most commonly used. The Python programming language provides arithmetic operators that perform addition, subtraction, multiplication, and division. It works the same as basic mathema

There are seven arithmetic operators we can use to perform different mathematical operations, such as:

1. + (Addition)
2. - (Subtraction)
3. \* (Multiplication)
4. / (Division)
5. // Floor division)
6. ℅ (Modulus)
7. \*\* (Exponentiation)

### Addition operator +

It adds two or more operands and gives their sum as a result

**Example**

x = 10

y = 40

**print**(x + y)

**or-**

name = "Madhuri"

surname = "Adat"

print(surname + " " + name)

Subtraction **-**

Use to subtracts the second value from the first value and gives the difference between them.

**Example**

x = 10

y = 40

**print**(y - x)

### **Multiplication \***

Multiply two operands. In simple terms, it is used to multiplies two or more values and gives their product as a result. The multiplication operator is denoted by a \* symbol.

**Example**

x = 2

y = 4

z = 5

**print**(x \* y)

### **Division /**

Divide the left operand (dividend) by the right one (divisor) and provide the result (quotient ) in a float value. The division operator is denoted by a / symbol. **Example**

x = 2

y = 4

z = 8

**print**(y / x)

# Output 2.0

**print**(z / y / x)

# Output 1.0

### **Floor division //**

Floor division returns the quotient (the result of division) in which the digits after the decimal point are removed

Example

x = 13

y = 2

print(x // y)

### **Modulus ℅**

The remainder of the division of left operand by the right. The modulus operator is denoted by a % symbol.

**Example**

x = 15

y = 4

**print**(x % y)

### **Exponent \*\***

Using exponent operator left operand raised to the power of right. The exponentiation operator is denoted by a double asterisk \*\* symbol. You can use it as a shortcut to calculate the exponential value

**Example**

num = 2

# 2\*2

**print**(num \*\* 2)

# Output 4

# 2\*2\*2

**print**(num \*\* 3)

## 2.Relational (comparison) operators-

## Relational operators are also called comparison operators. It performs a comparison between two values.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| > (Greater than) | It returns True if the left operand is greater than the right | x > y  result is True |
| < (Less than) | It returns True if the left operand is less than the right | x < y  result is False |
| == (Equal to) | It returns True if both operands are equal | x == y  result is False |
| != (Not equal to) | It returns True if both operands are equal | x != y  result is True |
| >= (Greater than or equal to) | It returns True if the left operand is greater than or equal to the right | x >= y  result is True |
| <= (Less than or equal to) | It returns True if the left operand is less than or equal to the right | x <= y  result is False |

## 3.Assignment operators-

## In Python, Assignment operators are used to assigning value to the variable. Assign operator is denoted by = symbol. For example, name = "Jessa" here, we have assigned the string literal ‘Jessa’ to a variable name

| **Operator** | **Meaning** | **Equivalent** |
| --- | --- | --- |
| = (Assign) | a=5Assign 5 to variable a | a = 5 |
| += (Add and assign) | a+=5Add 5 to a and assign it as a new value to a | a = a+5 |
| -= (Subtract and assign) | a-=5Subtract 5 from variable a and assign it as a new value to a | a = a-5 |
| \*= (Multiply and assign) | a\*=5Multiply variable a by 5 and assign it as a new value to a | a = a\*5 |
| /= (Divide and assign) | a/=5Divide variable a by 5 and assign a new value to a | a = a/5 |
| %= (Modulus and assign) | a%=5Performs modulus on two values and assigns it as a new value to a | a = a%5 |
| \*\*= (Exponentiation and assign) | a\*\*=5Multiply a five times and assigns the result to a | a = a\*\*5 |
| //= (Floor-divide and assign) | a//=5Floor-divide a by 5 and assigns the result to a | a = a//5 |

.

**Logical operators**

Logical operators are useful when checking a condition is true or not. Python has three logical operators. All logical operator returns a boolean value True or False depending on the condition in which it is used.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| and (Logical and) | True if both the operands are True | a and b |
| or (Logical or) | True if either of the operands is True | a or b |
| not (Logical not) | True if the operand is False | not a |

Example= and

a = 2

b = 4

# Logical and

**if** a > 0 **and** b > 0:

# both conditions are true

**print**(a \* b)

**else**:

**print**("Do nothing")

Example=or

**if** a > 0 **or** b < 0:

# at least one expression is true so conditions is true

**print**(a + b) # 6

**else**:

**print**("Do nothing")

**Example=Logical not**

**if** **not** a:

# a is True so expression is False

**print**(a)

**else**:

**print**("Do nothing")

## Membership operators

Python’s membership operators are used to check for membership of objects in sequence, such as string, list, tuple. It checks whether the given value or variable is present in a given sequence. If present, it will return True else False.

### **In operator**

It returns a result as True if it finds a given object in the sequence. Otherwise, it returns False.

**Example**

my\_list = [11, 15, 21, 29, 50, 70]

number = 15

**if** number **in** my\_list:

**print**("number is present")

**else**:

**print**("number is not present")

### Not in operator

It returns True if the object is not present in a given sequence. Otherwise, it returns False

**Example**

my\_tuple = (11, 15, 21, 29, 50, 70)

number = 35

**if** number **not** **in** my\_tuple:

**print**("number is not present")

**else**:

**print**("number is present")

## Identity operators

Use the Identity operator to check whether the value of two variables is the same or not

Python has 2 identity operators is and is not.

### **is operator**

The is operator returns Boolean True or False. It Return True if the memory address first value is equal to the second value. Otherwise, it returns False.

**Example**

x = 10

y = 11

z = 10

**print**(x **is** y) # it compare memory address of x and y

**print**(x **is** z) # it compare memory address of x and z

### **is not operator**

The is not the operator returns boolean values either True or False. It Return True if the first value is not equal to the second value. Otherwise, it returns False.

**Example**

x = 10

y = 11

z = 10

**print**(x **is** **not** y) # it campare memory address of x and y

**print**(x **is** **not** z) # it campare memory address of x and z

## Bitwise Operators

The bitwise operator operates on values bit by bit, so it’s called **bitwise**. It always returns the result in decimal format. Python has 6 bitwise operators listed below.

1. & Bitwise and
2. | Bitwise or
3. ^ Bitwise xor
4. ~ Bitwise 1’s complement
5. << Bitwise left-shift
6. >> Bitwise right-shift

### **Bitwise and**

It performs **logical AND** operation on the integer value after converting an integer to a binary value and gives the result as a decimal value. It returns True only if both operands are True. Otherwise, it returns False.

**Example**

a = 7

b = 4

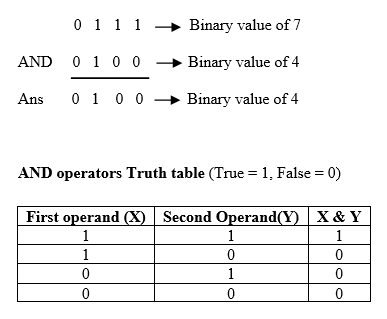
c = 5

**print**(a & b)

**print**(a & c)

**print**(b & c)

Following diagram shows AND operator evaluation.



### **Bitwise or**

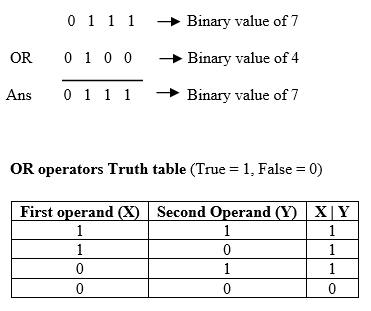
 It returns False only if both operands are True. Otherwise, it returns True.

**Example**

a = 7

b = 4

**print**(a or b)



Python bitwise OR

### **Bitwise xor ^**

It performs Logical XOR ^ operation on the binary value of a integer and gives the result as a decimal value.

**Example**: –

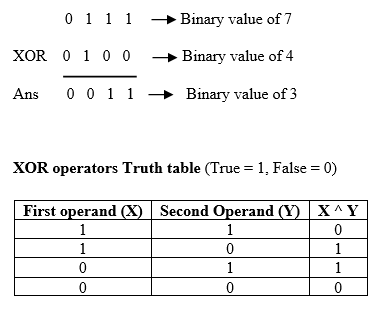
a = 7

b = 4

c = 5

**print**(a ^ c)

**print**(b ^ c)

Python bitwise XOR

### **Bitwise 1’s complement ~**

It performs 1’s complement operation. It invert each bit of binary value and returns the bitwise negation of a value as a result.

**Example**

a = 7

b = 4

c = 3

**print**(~a, ~b, ~c)

### **Bitwise left-shift <<**

The left-shift << operator performs a shifting bit of value by a given number of the place and fills 0’s to new positions.

**Example**: –

**print**(4 << 2)

# Output 16

**print**(5 << 3)

# Output 40

### **Bitwise right-shift >>**

The left-shift >> operator performs shifting a bit of value to the right by a given number of places. Here some bits are lost.

**print**(4 >> 2)

# Output

**print**(5 >> 2)

# Output

## Python Operators Precedence

In Python, operator precedence and associativity play an essential role in solving the expression. An expression is the combination of variables and operators that evaluate based on operator precedence.

We must know what the precedence (priority) of that operator is and how they will evaluate down to a single value.

| **Precedence level** | **Operator** | **Meaning** |
| --- | --- | --- |
| 1 (Highest) | () | Parenthesis |
| 2 | \*\* | Exponent |
| 3 | +x, -x ,~x | Unary plus, Unary Minus, Bitwise negation |
| 4 | \*, /, //, % | Multiplication, Division, Floor division, Modulus |
| 5 | +, - | Addition, Subtraction |
| 6 | <<, >> | Bitwise shift operator |
| 7 | & | Bitwise AND |
| 8 | ^ | Bitwise XOR |
| 9 | | | Bitwise OR |
| 10 | ==, !=, >, >=, <, <= | Comparison |
| 11 | is, is not, in, not in | Identity, Membership |
| 12 | not | Logical NOT |
| 13 | and | Logical AND |
| 14 (Lowest) | or | Logical OR |

**Control Flow Statements**

The flow control statements are divided into **three categories**

1. Conditional statements
2. Iterative statements.
3. Transfer statements

### 1.Conditional statements-

In Python, condition statements act depending on whether a given condition is true or false.

There are three types of conditional statements.

1. if statement
2. if-else
3. if-elif-else
4. nested if-els

### 2.Iterative statements

In Python, iterative statements allow us to execute a block of code repeatedly as long as the condition is True. We also call it a loop statements

1. [for loop](https://pynative.com/python-for-loop/)
2. [while loop](https://pynative.com/python-while-loop/)

### 3.Transfer statements

In Python, [transfer statements](https://pynative.com/python-break-continue-pass/) are used to alter the program’s way of execution in a certain manner. For this purpose, we use three types of transfer statements.

1. [break statement](https://pynative.com/python-break-continue-pass/#h-break-statement-in-python)
2. [continue statement](https://pynative.com/python-break-continue-pass/#h-continue-statement-in-python)
3. Pass statements

## If statement-

## if the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and The controller moves to the next line

**Syntax of the if statement**

**if** condition:

statement 1

statement 2

statement n

**Example**

number = 6

**if** number > 5:

# Calculate square

**print**(number \* number)

**print**('Next lines of code')

### **If – else statement-**

### The if-else statement checks the condition and executes the if block of code when the condition is True, and if the condition is False, it will execute the else block of code.

**Syntax of the if-else statement**

**if** condition:

statement 1

**else**:

statement 2

**Example**

number = int(input(“Enter the number:”)

**if** number > 5:

**print**(“number is grater then 5”)

else:

print(“number is smaller than 5”)

### **Chain multiple if statement in Python-**

In Python, the if-elif-else condition statement has an elif blocks to chain multiple conditions one after another. This is useful when you need to check multiple conditions.

**Syntax of the if-elif-else statement:**

**if** condition-1:

statement 1

**elif** condition-2:

stetement 2

**elif** condition-3:

stetement 3

...

**else**:

statement

**Example**

def user\_check(choice):

if choice == 1:

print("Admin")

elif choice == 2:

print("Editor")

elif choice == 3:

print("Guest")

else:

print("Wrong entry")

user\_check(1)

user\_check(2)

user\_check(3)

user\_check(4)

### **Nested if-else statement-**

In Python, the nested if-else statement is an if statement inside another if-else statement. It is allowed in Python to put any number of if statements in another if statement.

**Syntax of the nested-if-else:**

**if** conditon\_outer:

**if** condition\_inner:

statement of inner **if**

**else**:

statement of inner **else**:

statement ot outer **if**

**else**:

Outer **else**

num1 = **int**(**input**('Enter first number '))

num2 = **int**(**input**('Enter second number '))

**if** num1 >= num2:

**if** num1 == num2:

**print**(num1, 'and', num2, 'are equal')

**else**:

**print**(num1, 'is greater than', num2)

**else**:

**print**(num1, 'is smaller than', num2)

### **Single statement suites**

Whenever we write a block of code with multiple if statements, indentation plays an important role. But sometimes, there is a situation where the block contains only a single line statement.

1.exmple for if-else:

a=10

print(True ) if a<15 else print(False)

2.exmple-

a=23

b=31

c=14

max=(a if a>b and a>c else b if b>c else c)

print(max)

## for loop-

## Using for loop, we can iterate any sequence or iterable variable. The sequence can be string, [list](https://pynative.com/python-lists/), [dictionary](https://pynative.com/python-dictionaries/), [set](https://pynative.com/python-sets/), or [tuple](https://pynative.com/python-tuples/).

**Syntax of for loop:**

**for** element **in** sequence:

body of **for** loop

**for** i **in** **range**(1, 11):

## print(i)

## While loop in Python

In Python, The while loop statement repeatedly executes a code block while a particular condition is true.

**Syntax of while-loop**

**while** condition :

body of **while** loop

**Example to calculate the sum of first ten numbers**

num = 10

**sum** = 0

i = 1

**while** i <= num:

**sum** = **sum** + i

i = i + 1

**print**("Sum of first 10 number is:", **sum**)

## Break Statement in Python

## **The**[break statement](https://pynative.com/python-break-continue-pass/#h-break-statement-in-python) is used inside the loop to exit out of the loop. It is useful when we want to terminate the loop as soon as the condition is fulfilled instead of doing the remaining iterations

**Example of using a break statement**

**for** num **in** **range**(10):

**if** num > 5:

**print**("stop processing.")

**break**

**print**(num)

## Continue statement in python

## The [continue statement](https://pynative.com/python-break-continue-pass/#h-continue-statement-in-python) is used to skip the current iteration and continue with the next iteration

**Example of a continue statement**

**for** num **in** **range**(3, 8):

**if** num == 5:

**continue**

**else**:

**print**(num)

## Pass statement in Python

The pass is the keyword In Python, which won’t do anything. Sometimes there is a situation in programming where we need to define a syntactically empty block. We can define that block with the pass keyword.

**Example**

months = ['January', 'June', 'March', 'April']

**for** mon **in** months:

**pass**

**print**(months)