Python Operators

In Python, operators are special symbols that designate that some sort of computation should be performed. The values that an operator acts on are called **operands**.

Here is an example:

```
>>>
>>> a = 10
>>> b = 20
>>> print(a + b)
>>> Output = 30
```

In this case, the + operator adds the operands a and b together. An operand can be either a literal value or a variable that references an object :

```
>>>
>>> a = 10
>>> b = 20
>>> a + b - 5
>>> Output = 25
```

A sequence of operands and operators, like a + b - 5, is called an **expression**. Python supports many operators for combining data objects into expressions.

Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.

Python Operator falls into 7 categories:

Arithmetic operators

Comparison operators

Assignment Operators

Logical Operators

Bitwise Operators

Membership Operators

Identity Operators

Arithmetic operators: Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division

The following table lists the arithmetic operators supported by Python:

Operator	**Example**	**Meaning**	**Result**
+ (unary)	+a	**Unary Positive**	a - In other words, it doesn't really do anything. It mostly exists for the sake of completeness, to complement **Unary Negation**.
+ (binary)	a + b	**Addition**	Sum of a and b
- (unary)	-a	**Unary Negation**	Value equal to a but opposite in sign
- (binary)	a - b	**Subtraction**	b subtracted from a
*	a * b	**Multiplication**	Product of a and b
/	a / b	**Division**	Quotient when a is divided by b. The result always has type float.
%	a % b	**Modulo**	Remainder when a is divided by b
//	a // b	**Floor Division** (also called **Integer Division**)	Quotient when a is divided by b, rounded to the next smallest whole number
**	a ** b	**Exponentiation**	a raised to the power of b

- ✓ The result of standard division (/) is always a float, even if the dividend is evenly divisible by the divisor
- ✓ When the result of floor division (//) is positive, it is as though the fractional portion is truncated off, leaving only the integer portion. When the result is negative, the result is rounded down to the next smallest (greater negative) integer:
- ✓ Note, by the way, that in a REPL session, you can display the value of an expression by just typing it in at the >>> prompt without print(), the same as you can with a literal value or variable

Relational Operators : Comparison or Relational operators carries out the comparison between operands. They tell us whether an operand is greater than the other, lesser, equal, or a combination of those.

Operator	Example	Meaning	Result
==	a == b	Equal to	True if the value of a is equal to the value of b False otherwise
!=	a != b	Not equal to	True if a is not equal to b False otherwise
<	a < b	Less than	True if a is less than b False otherwise
<=	a <= b	Less than or equal to	True if a is less than or equal to b False otherwise
>	a > b	Greater than	True if a is greater than b False otherwise
>=	a >= b	Greater than or equal to	True if a is greater than or equal to b False otherwise

Comparison operators are typically used in Boolean contexts like conditional and loop statements to direct program flow, as you will see later

Equality Comparison on Floating-Point Values :

Recall from the earlier discussion of floating-point numbers that the value stored internally for a float object may not be precisely what you'd think it would be. For that reason, it is poor practice to compare floating-point values for exact equality.

Consider this example:

```
>>>
>>> x = 1.1 + 2.2
>>> x == 3.3
>>> Output : False
```

The internal representations of the addition operands are not exactly equal to 1.1 and 2.2, so you cannot rely on x to compare exactly to 3.3.

The preferred way to determine whether two floating-point values are "equal" is to compute whether they are close to one another, given some tolerance. Take a look at this example:

```
>>>
>>> tolerance = 0.00001
>>> x = 1.1 + 2.2
>>> abs(x - 3.3) < tolerance
>>> Output : True
```

abs() returns absolute value. If the absolute value of the difference between the two numbers is less than the specified tolerance, they are close enough to one another to be considered equal.

Assignment Operators : Assignment operators are used in Python to assign values to variables.

- ✓ `a = 5` is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- ✓ There are various compound operators in Python like `a += 5` that adds to the variable and later assigns the same. It is equivalent to `a = a + 5`
- ✓ Python supports a shorthand augmented assignment notation for the arithmetic and bitwise operators which are as follows

OPERATOR	DESCRIPTION	SYNTAX	
=	Assign value of right side of expression to left side operand	x = y + z	
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b	a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b	a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b	a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b	a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b	a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b	a=a//b

**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b	a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b	a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b	a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b	a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b	a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b	a= a << b

Logical Operators : These are conjunctions that you can use to combine more than one condition.We have three Python logical operator – **and, or, and not**

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

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.

Bitwise operators : Bitwise operators act on operands as if they were strings of binary digits.

They operate bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Result	Example
&	bitwise AND	Each bit position in the result is the logical AND of the bits in the corresponding position of the operands. (1 if both are 1, otherwise 0.)	x & y = 0 (0000 0000)
|	bitwise OR	Each bit position in the result is the logical OR of the bits in the corresponding position of the operands. (1 if either is 1, otherwise 0.)	x | y = 14 (0000 1110)
~	bitwise negation	Each bit position in the result is the logical negation of the bit in the corresponding position of the operand. (1 if 0, 0 if 1.)	~x = -11 (1111 0101)
^	bitwise XOR (exclusive OR)	Each bit position in the result is the logical XOR of the bits in the corresponding position of the operands. (1 if the bits in the operands are different, 0 if they are the same.)	x ^ y = 14 (0000 1110)
>>	Shift right n places	Each bit is shifted right n places.	x >> 2 = 2 (0000 0010)
<<	Shift left n places	Each bit is shifted left n places.	x << 2 = 40 (0010 1000)

Identity operators : is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Membership operators : in and not in are the membership operators in Python.
They are used to test whether a value or variable is found in a sequence
(string, list, tuple, set and dictionary).
In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Operator Precedence

The precedence of the operators is important to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in python is given below.

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