Python Operators: Arithmetic, Assignment, Comparison, Logical, Identity, Membership, Bitwise

Operators are special symbols that perform some operation on operands and returns the result. For example, `5 + 6` is an expression where `+` is an operator that performs arithmetic add operation on numeric left operand `5` and the right side operand `6` and returns a sum of two operands as a result.

Python includes the operator module that includes underlying methods for each operator. For example, the `+` operator calls the `operator.add(a,b)` method.

Example: Operator Methods

```
>>> 5 + 6
11
>>> import operator
>>> operator.add(5, 6)
11
>>> operator.__add__(5, 6)
11
```

Above, expression `5 + 6` is equivalent to the expression `operator.add(5, 6)` and `operator.__add__(5, 6)`. Many function names are those used for special methods, without the double underscores (dunder methods). For backward compatibility, many of these have functions with the double underscores kept.

Python includes the following categories of operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Identity Operators
- Membership Test Operators
- Bitwise Operators

Arithmetic Operators

Arithmetic operators perform the common mathematical operation on the numeric operands.

The arithmetic operators return the type of result depends on the type of operands, as below.

1. If either operand is a complex number, the result is converted to complex;
2. If either operand is a floating point number, the result is converted to floating point;
3. If both operands are integers, then the result is an integer and no conversion is needed.

The following table lists all the arithmetic operators in Python:

Operation	Operator	Function	Example in Python Shell

Operation	Operator	Function	Example in Python Shell

Addition: Sum of two operands	+	operator.add(a,b)	`>>> x = 5; y = 6`
			`>>> x + y`
			`11`
			`>>> import operator`
			`>>> operator.add(5,6)`
			`11`

Subtraction: Left operand minus right operand	-	operator.sub(a,b)	`>>> x = 10; y = 5`
			`>>> x - y`
			`5`
			`>>> import operator`
			`>>> operator.sub(10, 5)`
			`5`

Multiplication	*	operator.mul(a,b)	`>>> x = 5; y = 6`
			`>>> x * y`
			`30`
			`>>> import operator`
			`>>> operator.mul(5,6)`
			`30`

Exponentiation: Left operand raised to the power of right	**	operator.pow(a,b)	`>>> x = 2; y = 3`
			`>>> x ** y`
			`8`
			`>>> import operator`
			`>>> operator.pow(2, 3)`
			`8`

Division	/	operator.truediv(a,b)	`>>> x = 6; y = 3`
			`>>> x / y`
			`2`
			`>>> import operator`
			`>>> operator.truediv(6, 3)`
			`2`

Floor division: equivilant to `math.floor(a/b)`	//	operator.floordiv(a,b)	`>>> x = 6; y = 5`
			`>>> x // y`
			`1`
			`>>> import operator`
			`>>> operator.floordiv(6,5)`
			`1`

Modulus: Reminder of `a/b`	%	operator.mod(a, b)	`>>> x = 11; y = 3`
			`>>> x % y`
			`12`
			`>>> import operator`
			`>>> operator.mod(11, 3)`
			`2`

Assignment Operators

The assignment operators are used to assign values to variables. The following table lists all the arithmetic operators in Python:

Operator	Function	Example in Python Shell

=		`>>> x = 5;`
		`>>> x`
		`5`

+=	operator.iadd(a,b)	`>>> x = 5`
		`>>> x += 5`
		`10`
		`>>> import operator`
		`>>> x = operator.iadd(5, 5)`
		`10`

-=	operator.isub(a,b)	`>>> x = 5`
		`>>> x -= 2`
		`3`
		`>>> import operator`
		`>>> x = operator.isub(5,2)`

*=	operator.imul(a,b)	`>>> x = 2`
		`>>> x *= 3`
		`6`
		`>>> import operator`
		`>>> x = operator.imul(2, 3)`

/=	operator.itruediv(a,b)	`>>> x = 6`
		`>>> x /= 3`
		`2`
		`>>> import operator`
		`>>> x = operator.itruediv(6, 3)`

//=	operator.ifloordiv(a,b)	`>>> x = 6`
		`>>> x //= 5`
		`1`
		`>>> import operator`
		`>>> operator.ifloordiv(6,5)`

Operator	Function	Example in Python Shell

%=	operator.imod(a, b)	`>>> x = 11`
		`>>> x %= 3`
		`2`
		`>>> import operator`
		`>>> operator.imod(11, 3)`
		`2`

&=	operator.iand(a, b)	`>>> x = 11`
		`>>> x &= 3`
		`1`
		`>>> import operator`
		`>>> operator.iand(11, 3)`
		`1`

|=	operator.ior(a, b)	`>>> x = 3`
		`>>> x |= 4`
		`7`
		`>>> import operator`
		`>>> operator.mod(3, 4)`
		`7`

^=	operator.ixor(a, b)	`>>> x = 5`
		`>>> x ^= 2`
		`7`
		`>>> import operator`
		`>>> operator.ixor(5, 2)`
		`7`

>>=	operator.irshift(a, b)	`>>> x = 5`
		`>>> x >>= 2`
		`1`
		`>>> import operator`
		`>>> operator.irshift(5, 2)`
		`1`

<<=	operator.ilshift(a, b)	`>>> x = 5`
		`>>> x <<= 2`
		`20`
		`>>> import operator`
		`>>> operator.ilshift(5, 2)`
		`20`

Comparison Operators

The comparison operators compare two operands and return a boolean either True or False. The following table lists comparison operators in Python.

Operator	Function	Description	Example in Python Shell

Operator	Function	Description	Example in Python Shell

>	operator.gt(a,b)	True if the left operand is higher than the right one	>>> x = 5; y = 6
			>>> x > y
			False
			>>> import operator
			>>> operator.gt(5,6)
			False

<	operator.lt(a,b)	True if the left operand is lower than right one	>>> x = 5; y = 6
			>>> x < y
			True
			>>> import operator
			>>> operator.add(5,6)
			True

==	operator.eq(a,b)	True if the operands are equal	>>> x = 5; y = 6
			>>> x == y
			False
			>>> import operator
			>>> operator.eq(5,6)
			False

!=	operator.ne(a,b)	True if the operands are not equal	>>> x = 5; y = 6
			>>> x != y
			True
			>>> import operator
			>>> operator.ne(5,6)
			True

>=	operator.ge(a,b)	True if the left operand is higher than or equal to the right one	>>> x = 5; y = 6
			>>> x >= y
			False
			>>> import operator
			>>> operator.ge(5,6)
			False

<=	operator.le(a,b)	True if the left operand is lower than or equal to the right one	>>> x = 5; y = 6
			>>> x <= y
			True
			>>> import operator
			>>> operator.le(5,6)
			True

Logical Operators

The logical operators are used to combine two boolean expressions. The logical operations are generally applicable to all objects, and support truth tests, identity tests, and boolean operations.

Operator	Description	Example

and	True if both are true	>>> x = 5; y = 6
		>>> x > 1 and y <10
		True

Operator	Description	Example

or	True if at least one is true	`>>> x = 5; y = 6`
		`>>> x > 6 or y <10`
		`True`

not	Returns True if an expression evalutes to false and vice-versa	`>>> x = 5`
		`>>> not x > 1`
		`False`

Identity Operators

The identity operators check whether the two objects have the same id value e.i. both the objects point to the same memory location.

Operator	Function	Description	Example in Python Shell

is	operator.is_(a,b)	True if both are true	`>>> x = 5; y = 6`
			`>>> x is y`
			`False`
			`>>> import operator`
			`>>> operator.is_(x,y)`
			`False`

is not	operator.is_not(a,b)	True if at least one is true	`>>> x = 5; y = 6`
			`>>> x is not y`
			`True`
			`>>> import operator`
			`>>> operator.is_not(x, y)`
			`True`

Membership Test Operators

The membership test operators `in` and `not in` test whether the sequence has a given item or not. For the string and bytes types, `x in y` is True if and only if `x` is a substring of `y`.

Operator	Function	Description	Example in Python Shell

in	operator.contains(a,b)	Returns True if the sequence contains the specified item else returns False.	`>>> nums = [1,2,3,4,5]`
			`>>> 1 in nums`
			`True`
			`>>> 10 in nums`
			`False`
			`>>> 'str' in 'string'`
			`True`
			`>>> import operator`
			`>>> operator.contains(nums, 2)`
			`True`

Operator	Function	Description	Example in Python Shell

not in	not operator.contains(a,b)	Returns True if the sequence does not contains the specified item, else returns False.	`>>> nums = [1,2,3,4,5]`
			`>>> 1 not in nums`
			`False`
			`>>> 10 not in nums`
			`True`
			`>>> 'str' not in 'string'`
			`False`
			`>>> import operator`
			`>>> not operator.contains(nums, 2)`
			`False`

Bitwise Operators

Bitwise operators perform operations on binary operands.

Operator	Function	Description	Example in Python Shell

&	operator.and_(a,b)	Sets each bit to 1 if both bits are 1.	`>>> x=5; y=10`
			`>>> z=x & y`
			`>>> z`
			`0`
			`>>> import operator`
			`>>> operator.and_(x, y)`
			`0`

|	operator.or_(a,b)	Sets each bit to 1 if one of two bits is 1.	`>>> x=5; y=10`
			`>>> z=x | y`
			`>>> z`
			`15`
			`>>> import operator`
			`>>> operator.or_(x, y)`
			`15`

^	operator.xor(a,b)	Sets each bit to 1 if only one of two bits is 1.	`>>> x=5; y=10`
			`>>> z=x ^ y`
			`>>> z`
			`15`
			`>>> import operator`
			`>>> operator.xor(x, y)`
			`15`

~	operator.invert(a)	Inverts all the bits.	`>>> x=5`
			`>>> ~x`
			`-6`
			`>>> import operator`
			`>>> operator.invert(x)`
			`-6`

<<	operator.lshift(a,b)	Shift left by pushing zeros in from the right and let the leftmost bits fall off.	`>>> x=5`
			`>>> x<<2`
			`20`
			`>>> import operator`
			`>>> operator.lshift(x,2)`
			`20`

Operator	Function	Description	Example in Python Shell

>>	operator.rshift(a,b)	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.	>>> x=5
			>>> x>>2
			1
			>>> import operator
			>>> operator.rshift(x,2)
			1

Related Articles

- Compare strings in Python
- Convert file data to list
- Convert User Input to a Number
- Convert String to Datetime in Python
- How to call external commands in Python?
- How to count the occurrences of a list item?
- How to flatten list in Python?
- How to merge dictionaries in Python?
- How to pass value by reference in Python?
- Remove duplicate items from list in Python
- More Python articles

‹ Previous Next ›

feedback@tutorialsteacher.com

- Web API
- C#
- LINQ
- Python
- SQL Server
- MongoDB
- Entity Framework
- Node.js
- Angular 2
- D3.js
- Sass
- Https (SSL)
- AngularJS 1

E-mail list

Subscribe to TutorialsTeacher email list and get latest updates, tips & tricks on C#, .Net, JavaScript, jQuery, AngularJS, Node.js to your inbox.

Email address GO

We respect your privacy.