Python Bootcamp & Masterclass

arithmetic operators

gknxt

Operators

Arithmetic +

Comparision >

Assignment =

Logical or

Bitwise &

Membership in

Identity is

gk nxt

Arithmetic Operators

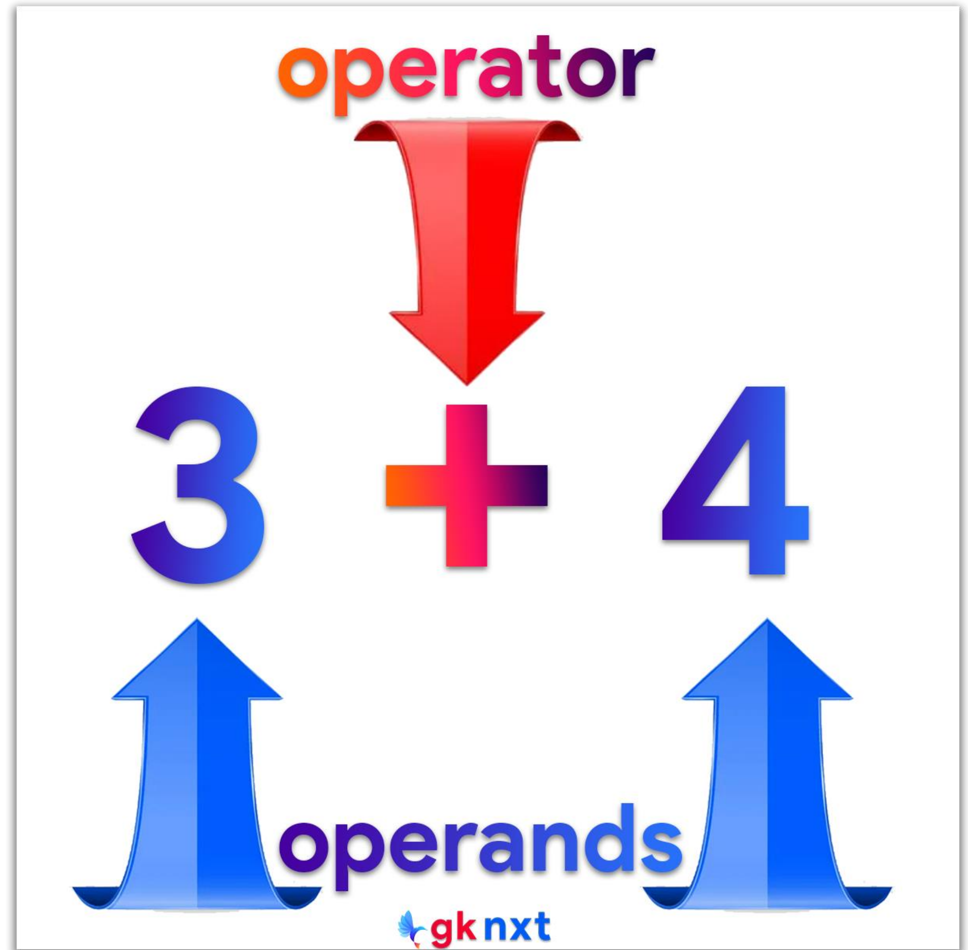
Expression	Result
3 + 2	5
3 - 2	1
3 * 2	6
3 / 2	1.5
3 // 2	1
3 ** 2	9
3 % 2	1

gk nxt

operators

An expression is a combination of variables, operators, objects, parentheses and calls to functions that Python can compute or evaluate to return the result.

Operators are special symbols that designate that some sort of computation should be performed. The values that an operator acts on are called operands.



gk nxt

+ (unary) and - (unary)

Unary positive acts on one operand and keeps the positive sign of the operand positive and negative sign of the operand negative (so, its effect is practically nothing)

Unary negation acts on one operand and switches the sign of the operand (if the operand is positive, unary negation makes it negative and if the operand is negative, unary negation makes it positive)

```
a = 4
+a
++a
```

```
4

4
```

```
b = 5
-b
--b
---b
-+b
-+--+b
```

```
-5

5

-5

-5

5
```

gknxt

+ operator adds the two operands if both are of numeric type. It concatenates if the operands are strings or lists or tuples. It is an overloaded operator and will actually call **__add__** (double underscore add double underscore) magic method under the hood (so, based on the operands, it adds/concatenates/raises error)

```
4.4 + 3                       # + as addition operator
4.4.__add__(3)                # + internally calls the magic method __add__
4 + True                      # + as addition operator (True evaluates as 1)
7.7 + False                   # + as addition operator (False evaluates as 0)
True + False                  # + as addition operator
```

```
7.4

7.4

5

7.7

1
```

```
'a' + 'b' + '7'              # + as concatenation operator (concatenation of strings)
'a'.__add__('b').__add__('7')  # + internally calls the magic method __add__
[1, 2, 3] + ['a', 'b']       # + as concatenation operator (concatenation of lists)
(1, 2, 3) + ('a', 'b')       # + as concatenation operator (concatenation of tuples)
#{1, 2, 3} + {'a', 'b'}      # sets cannot be cocatenated with + as __add__ is not implemented for set object
```

```
'ab7'

'ab7'

[1, 2, 3, 'a', 'b']

(1, 2, 3, 'a', 'b')
```

gk nxt

`*` operator multiplies the two operands if both are of numeric type. It is an overloaded operator and will actually call `__mul__` and if it fails, calls `__rmul__` (difference between `x.__mul__(y)` and `x.__rmul__(y)` is that the former calculates `x * y` whereas the latter calculates `y * x`)

`*` operator replicate the first operands if it is a string/list/tuple and second operand is an integer type (True and False are also of integer type)

`*` as unary operator can be used for packing/unpacking multiple objects into a single container

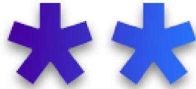
```python
4 * 2
3.5 * 4.2
(2 + 4j) * (3 + 5j)
True * 3                    # True evaluates as 1
False * 4                   # False evaluates as 0
```

```
8

14.700000000000001

(-14+22j)

3

0
```

```python
'Hello ' * 5                # string replication
'Hello '.__mul__(5)
[1, 2, 3] * 4               # list replication
[1, 2, 3].__mul__(4)
('a', 2, 3) * 3             # tuple replication
('a', 2, 3).__mul__(3)
[1, 2, 3] * False
('a', 2, 3) * True
```

```
'Hello Hello Hello Hello Hello '

'Hello Hello Hello Hello Hello '

[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

('a', 2, 3, 'a', 2, 3, 'a', 2, 3)

('a', 2, 3, 'a', 2, 3, 'a', 2, 3)

[]

('a', 2, 3)
```

gknxt



** as exponentiation operator raises the first operand to the power of the second operand if both are of numeric type. It is an overloaded operator and will actually call __pow__ magic method under the hood.

** as unary operator can be used for unpacking/merging multiple dictionaries into a single dictionary. It is also used for unpacking multiple keyword arguments.

```
2 ** 4          # 2 power 4 is same as 2*2*2*2
(2).__pow__(4)
1.2 ** -4.1
-3 * False
6 ** (2 + 4J)
True ** False
```

```
16

16

0.47354024139752743

0

(22.830375523713197+27.834761602108927j)

1
```

- operator (subtraction operator) subtracts the second operand from the first operand. It is an overloaded operator and will actually call `__sub__` magic method under the hood.

- operator (difference operator) returns a new set with elements in the the first operand that are not in the second operand .

```
4.4 - 3                            # - as subtraction operator
4.4.__sub__(3)                     # - internally calls the magic method __sub__
4 - True                           # - as subtraction operator (True evaluates as 1)
7.7 - False                        # - as subtraction operator (False evaluates as 0)
True - False                       # - as subtraction operator
```

```
1.4000000000000004

1.4000000000000004

3

7.7

1
```

```
{1, 2, 3} - {1, 2}                 # - as difference operator
```

```
{3}
```

gknxt

/

/ operator (division operator or float division operator) divides the first operand by the second operand if the operands are of numeric type. The result is always float if neither operand is complex and complex otherwise. The second operand cannot be zero/False. It will actually call `__truediv__` magic method under the hood.

```
float('inf') / 26
float('-inf') / 26
26 / float('inf')
26 / float('-inf')
26 / 0    # ZeroDivisionError
```

```
inf

-inf

0.0

-0.0

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_6488/3569365736.py in <module>
      3 26 / float('inf')
      4 26 / float('-inf')
----> 5 26 / 0            # ZeroDivisionError

ZeroDivisionError: division by zero
```

```
26 / 10
(26).__truediv__(10)
26 / 3j
2j / 26
3j / 2j
26 / True
False / True
```

```
2.6

2.6

-8.666666666666666j

0.07692307692307693j

(1.5+0j)

26.0

0.0
```

gknxt

// operator (floor division operator or integer division operator) divides the first operand by the second operand if the operands are of int, float or bool type. If the result is float, it will **round it down** to the next (small) integer, so the result is always int. The second operand cannot be zero/False. It will actually call `__floordiv__` magic method under the hood.

```
25 // 5
26 // 10
(26).__floordiv__(10)
# 26 // 3j          # TypeError: can't take floor of complex number
# 2j // 26          # TypeError: can't take floor of complex number
26 // True
False // True
-26 // 10
-26 // -10
```

```
5

2

2

26

0

-3

2
```

% operator (modulo operator) divides the dividend (first operand) by the divisor (second operand) and returns the reminder if the operands are of numeric type, but not complex. Result (remainder) takes the **sign of the divisor**. The divisor cannot be zero/False. It will actually call `__mod__` magic method under the hood.

`divmod()` internally uses modulo operator (`%`). It takes two parameters (dividend and divisor) and returns a tuple with the results of floor division and modulo (quotient, reminder)

```
# dividend % divisor will be computed in Python as
# (dividend - (divisor * (dividend//divisor)))
print("274 // 5 is:", 274//5)
print("274 % 5 is:", 274 % 5)
print("274 - (5 * 274//5) is:", 274 - (5 * (274//5)))
```

```
274 // 5 is: 54
274 % 5 is: 4
274 - (5 * 274//5) is: 4
```

```
divmod(274, 5)
```

```
(54, 4)
```

274 / 5

274 // 5 == 54			5	4	quotient
divisor	5	2	7	4	dividend
		2	5		
			2	4	
			2	0	
274 % 5 == 4				4	reminder

If dividend is even and divisor is 2, then the reminder (modulo) is `0`.

```
# if dividend is even and divisor is 2,
# then the reminder is 0
36 % 2
36.0 % 2
2 % 2
2.0 % 2
0 % 2
False % 2
-36 % 2
-36.0 % 2
-2 % 2
-2.0 % 2
```

```
0
0.0
0
0.0
0
0
0
0.0
0
0.0
```

```
# if dividend is even and divisor is 2,
# then the reminder is 0
36 % 2 == 0
36.0 % 2 == 0
2 % 2 == 0
2.0 % 2 == 0
0 % 2 == 0
False % 2 == 0
-36 % 2 == 0
-36.0 % 2 == 0
-2 % 2 == 0
-2.0 % 2 == 0
```

```
True
True
True
True
True
True
True
True
True
True
```

gknxt

If dividend is odd and divisor is 2, then the reminder (modulo) is **not** `0`.

```
# if dividend is odd and divisor is 2,
# then the reminder is not 0
37 % 2
37.1 % 2
1 % 2
1.0 % 2
-37 % 2
-37.1 % 2
-1 % 2
-1.0 % 2
```

```
1

1.1000000000000014

1

1.0

1

0.8999999999999986

1

1.0
```

```
# if dividend is odd and divisor is 2,
# then the reminder is not 0
37 % 2 == 0
37.1 % 2 == 0
1 % 2 == 0
1.0 % 2 == 0
-37 % 2 == 0
-37.1 % 2 == 0
-1 % 2 == 0
-1.0 % 2 == 0
```

```
False

False

False

False

False

False

False

False
```

gknxt

If dividend is int or equivalent and divisor is 1, then the reminder (modulo) is `0`.

```
# if dividend is int or equivalent and
# divisor is 1, then the reminder is 0
36 % 1
36.0 % 1
2 % 1
2.0 % 1
1 % 1
0 % 1
False % 1
-37 % 1
-37.0 % 1
-2 % 1
-2.0 % 1
-1 % 1
```

```
0

0.0

0

0.0

0

0

0

0

0.0

0

0.0

0
```

gk nxt

If dividend is int or equivalent and divisor is 1, then the reminder (modulo) is `0`.

```
# if both dividend and divisor have same
# absolute value, then the reminder == 0
2 % 2
2.0 % 2.0
-1 % -1
1 % 1
True % True
-1.0 % -1.0
1.0 % 1.0
-36 % -36
-36.0 % -36.0
-2 % -2
-2.0 % -2.0
```

```
0

0.0

0

0

0

-0.0

0.0

0

-0.0

0

-0.0
```

```
# if both dividend and divisor have same
# absolute value, then the reminder == 0
2 % -2
2.0 % -2.0
-1 % 1
1 % -1
-1.0 % 1.0
1.0 % -1.0
-36 % 36
-36.0 % 36.0
-2 % 2
-2.0 % 2.0
```

```
0

-0.0

0

0

0.0

-0.0

0

0.0

0

0.0
```

gknxt

If dividend is `0` or False, then the reminder (modulo) is `0`.

```
# if dividend is zero or False,
# then the reminder is 0
0 % 1
0 % 2
0 % 1.7
0 % -1
0 % -2
0 % -1.7
0 % True
0 % float('inf')
0 % float('-inf')
```

```
0

0

0.0

0

0

-0.0

0

0.0

-0.0
```

```
# if dividend is zero or False,
# then the reminder is 0
0.0 % 1
0.0 % 2
0.0 % 1.7
0.0 % -1
0.0 % -2
0.0 % -1.7
0.0 % True
0.0 % float('inf')
0.0 % float('-inf')
```

```
0.0

0.0

0.0

-0.0

-0.0

-0.0

0.0

0.0

-0.0
```

```
# if dividend is zero or False,
# then the reminder is 0
False % 1
False % 2
False % 1.7
False % -1
False % -2
False % -1.7
False % True
False % float('inf')
False % float('-inf')
```

```
0

0

0.0

0

0

-0.0

0

0.0

-0.0
```

gk nxt

If dividend is not negative and divisor is infinity, then the reminder is float of dividend.

If dividend is not negative and divisor is negative infinity, then the reminder is negative infinity.

```
# if dividend is not negative and divisor is infinity,
# then the reminder is float of dividend
37 % float('inf')
36.8 % float('inf')
2 % float('inf')
2.5 % float('inf')
1 % float('inf')
0 % float('inf')
```

```
37.0

36.8

2.0

2.5

1.0

0.0
```

```
# if dividend is not negative and divisor is negative
# infinity, then the reminder is negative infinity
37 % float('-inf')
36.8 % float('-inf')
2 % float('-inf')
2.5 % float('-inf')
36 % float('-inf')
1 % float('-inf')
```

```
-inf

-inf

-inf

-inf

-inf

-inf
```

If dividend is negative and divisor is infinity, then the reminder is infinity. If dividend is negative and divisor is negative infinity, then the reminder is same as dividend

```
# if dividend is negative and divisor is
# infinity then the reminder is infinity
-37 % float('inf')
-36.8 % float('inf')
-2 % float('inf')
-2.5 % float('inf')
-1 % float('inf')
```

```
inf

inf

inf

inf

inf
```

```
# if dividend is negative and divisor is negative
# infinity then the reminder is same as dividend
-37 % float('-inf')
-36.8 % float('-inf')
-2 % float('-inf')
-2.5 % float('-inf')
-1 % float('-inf')
-36 % float('-inf')
```

```
-37.0

-36.8

-2.0

-2.5

-1.0

-36.0
```

gknxt

If dividend is infinity or negative infinity, then the reminder is nan (not a number)

```
# if dividend is infinity or negative
# infinity,  then the reminder is nan
float('inf')  % 101
float('inf')  % 2
float('inf')  % -100
float('inf')  % -2
float('inf')  % float('inf')
float('inf')  % float('-inf')
```

```
nan
nan
nan
nan
nan
nan
```

```
# if dividend is infinity or negative
# infinity,  then the reminder is nan
float('-inf') % 1
float('-inf') % 2
float('-inf') % 206.9
float('-inf') % -1
float('-inf') % -202.7
float('-inf') % True
float('-inf') % float('inf')
float('-inf') % float('inf')
```

```
nan
nan
nan
nan
nan
nan
nan
nan
```

gknxt

Python's math module has `fmod()` method and it does the modulo operation differently. `math.fmod()` uses truncated division and takes the **sign of the dividend** for the remainder. Modulo operator with `decimal.Decimal` is guaranteed to maintain floating-point precision.

```
import math
37 % 5
math.fmod(37,5)

-37 % -5
math.fmod(-37,-5)
```

```
2

2.0

-2

-2.0
```

```
import math
37 % -5
math.fmod(37,-5)

-37 % 5
math.fmod(-37,5)
```

```
-3

2.0

3

-2.0
```

gknxt

% operator is an overloaded operator and can be used for string formatting. But it is recommended not to use this type of formatting as newer formatting options, **f-string** and **.format()**, provide flexibility and readability.

```
cust_name = 'Jane Doe'
cust_prod = ['credit', 'loan', 'account']
cust_status = ['pending', 'approved', 'denied']
print('Hello, %s,' %cust_name, 'your %s' %cust_prod[1], 'is %s' %cust_status[1])
```

```
Hello, Jane Doe, your loan is approved
```

Online Resources

For best python resources, please visit:

gknxt.com/python/

Python Bootcamp & Masterclass

Thank You

for your Rating & Review