Arithmetic and logical operators by precedence (higher precedence operators are evaluated before lower ones):

| | |
|---|---|
| `( )` | Parentheses |
| `**` | Exponentiation |
| `+x  -x  ~x` | Unary plus, unary minus, unary bitwise NOT |
| `*  /  //  %` | Multiplication, division, floor (integer) division, (integer) modulus |
| `+  -` | Addition and subtraction |
| `<<  >>` | Bitwise left and right shifts |
| `&  \|  ^` | Bitwise AND, OR, XOR |
| `==  !=  >  >=  <  <=`<br>`is  is not  in  not in` | Logical comparisons, identity, inequality, membership |
| `and  or  not` | Logical AND., OR, NOT |
| `:=` | Assignment within expression (walrus) |

Arithmetic and logical operator examples:

```
>>> 2**4
16
>>> 7/2
3.5
>>> 7//2
3
>>> 7%2
1
>>> (7//2)*2 + 7%2
7
>>> 13&3
1
>>> 13|3
15
>>> 13^1
12
```

```
>>> 5 > 2
True
>>> 5 < 2
False
>>> 3 > 3
False
>>> 3 >= 3
True
>>> 3 >= 2 or 1 > 17
True
>>> 3 >= 2 and 1 > 17
False
>>> not (5 > 2)
False
>>> not (2 > 5)
True
```

String operators:

| | |
|---|---|
| `+` | concatenation |
| `*` | repetition |

String operator examples:

```
>>> "cat" + "dog"
'catdog'
>>> "cat" * 3
'catcatcat'
```

```
>>> ("cat" + "dog") * 3
'catdogcatdogcatdog'
>>> "cat" * 0
''
```

Assignment operators:

| | |
|---|---|
| *variable* **=** *value* | assign arithmetic or string *value* to *variable* |
| `+=  -=  *=  /=  //=  %=` | add, subtract, multiply, divide, unary divide, or modulus variable by value |

Assignment operator examples (arithmetic and string):

```
>>> i = 7
>>> i
7
>>> i += 3
>>> i
10
```

```
>>> a = "cat"
>>> a
cat
>>> a += "dog"
>>> a
'catdog'
```

Print examples:

```
>>> a = 42
>>> print("a's value is", a)
a's value is 42
>>> print(42, 42+2, 42+4)
42 44 46
```

```
>>> print("3 + 5 is", 3+5)
3 + 5 is 8
>>> a = "john"
>>> print("hello", a)
hello john
```

Comparisons of variables:

```
>>> a = 5
>>> b = 2
>>> a == b
False
>>> a != b
True
>>> a > b
True
>>> a < b
False
```

```
>>> a = "cat"
>>> b = "dog"
>>> a == b
False
>>> a != b
True
>>> a > b
False
>>> b > a
True
```

Range() function returns an iterable object, which can be displayed as a list:

```
>>> range(8)
range(0, 8)
>>> list(range(8))
[0, 1, 2, 3, 4, 5, 6, 7]
>>> list(range(3,8))
[3, 4, 5, 6, 7]
>>> list(range(3,8,2))
[3, 5, 7]
```

"If statements" allow conditional execution of code:

```
>>> if 3 > 7:
...   print("greater")
... else:
...   print("not greater")
...
not greater
```

```
>>> if 3 > 3:
...   print("greater")
... elif 3 < 3:
...   print("less")
... else:
...   print("neither greater nor less")
...
neither greater nor less
```

“For loops” iterate code over iterable object or all items of a list (or other sequence) one at a time:

```
>>> for i in range(2,6):
...   print(i)
...
2
3
4
5
```

```
>>> for i in list(range(10,7,-1)):
...   print(i)
...
10
9
8
```

“While loops” repeat code until the conditional or logical expression is False:

```
>>> i = 2
>>> while i < 6:
...   print(i)
...   i += 1
...
2
3
4
5
```

```
>>> i = 10
>>> while i > 7:
...   print(i)
...   i = i - 1
...
10
9
8
```

“**break**” – causes loop to stop iterating early (and skip all remaining items or values)
“**continue**” – causes loop to immediately jump to top of next iteration (skipping further code in the current iteration)
“**else**” – (only in python!) runs when loop reaches the end of items or values (but not if “break” was executed)