Operators by precedence (higher precedence operators are evaluated before lower precedence operators):

`( )`	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
`^`	Bitwise XOR
`|`	Bitwise OR
`==  !=  >  >=  <  <=` `is  is not  in  not in`	Logical comparisons, identity, membership
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR
`:=`	Assignment within expression (walrus)

Examples (arithmetic and logical):

```
>>> 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
>>> 13 and 3
3
>>> 0 and 3
0
>>>
```

```
>>> 5 > 2
True
>>> 5 < 2
False
>>> 3 > 3
False
>>> 3 >= 3
True
>>> 3 >= 2 or 1 > 17
True
>>> 3 >= 2 and 1 > 17
False
>>> 7 > 3
True
>>> not (7 > 3)
False
>>> 3 > 7
False
>>> not (3 > 7)
True
>>>
```

Assignment operators (arithmetic and string):

```
>>> i=7
>>> i+=3
>>> i
10
>>> i*2
20
>>>
```

```
>>> a = "cat"
>>> a += "dog"
>>> a
'catdog'
>>> a*3
'catdogcatdogcatdog'
>>>
```

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]
>>>
```

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

```
>>> for i in range(4):
...  print(i)
...
0
1
2
3
>>>
```

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

"While loops" repeat until the conditional or logical expression is False:

```
>>> i = 2
>>> while i < 7:
...  print(i)
...  i += 1
...
2
3
4
5
6
>>>
```

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

Loop control:

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

More loop examples:

```
>>> i = 7
>>> while True:
...  print(i)
...  i += 2
...  if i > 12 and i%3==0:
...   break
...
7
9
11
13
>>> print("after the loop, i is", i)
after the loop, i is 15
>>>
```

What happened? We started by setting i to 7. Then we enter an "infinite loop" whose condition is "True" – this loop will continue running (infinitely) until a "break" statement is executed! For each iteration of the loop, we print the value of i. Then we add 2 to i. Then we test the value to i to see if it is both greater than 12 and divisible by 3 (using the mod 3 operator, looking for a remainder of 0). If it is, we break out of the infinite loop! When this happens, i is 15 (but had not yet been printed, so we do so afterwards)

```
>>> i=5
>>> while i<10:
...  i = i+1
...  print("thinking")
...  if i<8:
...   continue
...  print(i)
... else:
...  print("after the loop, i is", i)
...
thinking
thinking
thinking
8
thinking
9
thinking
10
after the loop, i is 10
>>>
```

What happened? We started by setting i to 5. Then we enter a conditional loop that will terminate when i is >= 10. For each iteration of the loop, we add 1 to i. Then we print "thinking" and test if i is < 8 – if it is, we continue back to the top of the loop without running any more code in the loop. Otherwise, we print i. Finally, when the loop is done, we print the value of i again.