Технологии программирования

The basic concepts of Python

Pakhomova K.I.,

Kononova N.V.

History

Author: Guido van Rossum

- 1994 – Python 1.0
- 2000 – Python 2.0
- 2008 – Python 3.0

The Special Aspects

- Easy way to…

1) debugging program code
2) write and correct
3) read
4) learn

- Free license (Python Software Foundation License)

But

Slow work = (In comparison with C++)

Why

Python is **Interpreted** high-level programming language

And

Python is **Dynamic typed**

The Zen of Python, by Tim Peters

import this

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Installation interpreter

- Python 3

https://www.python.org/downloads/

- PyCharm https://www.jetbrains.com/pycharm/
- VSCode https://code.visualstudio.com/
- MS Visual Studio

Module Installation

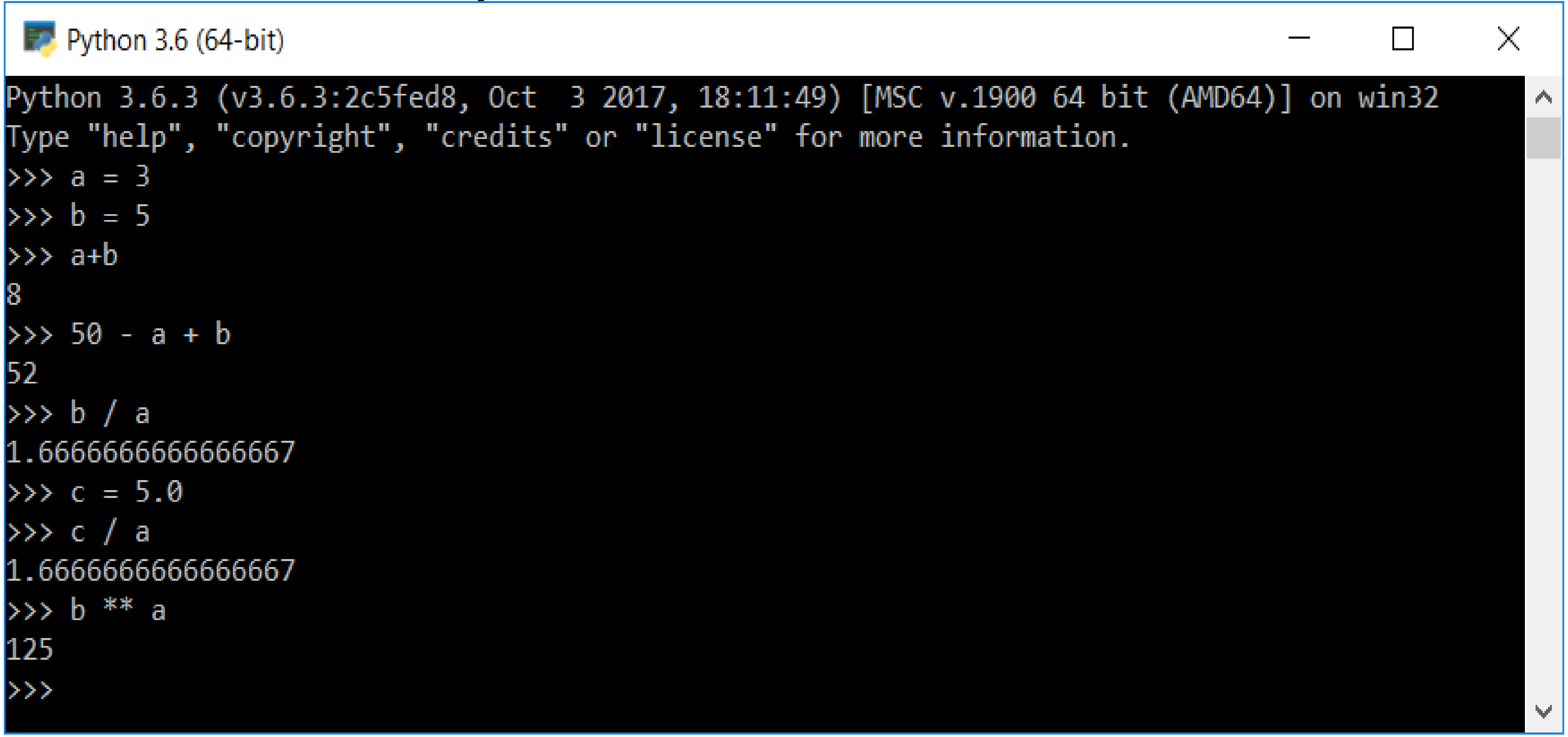
- easy_install package_name
- pip install package_name

(pip help, pip uninstall, pip list, pip install -U)

Help

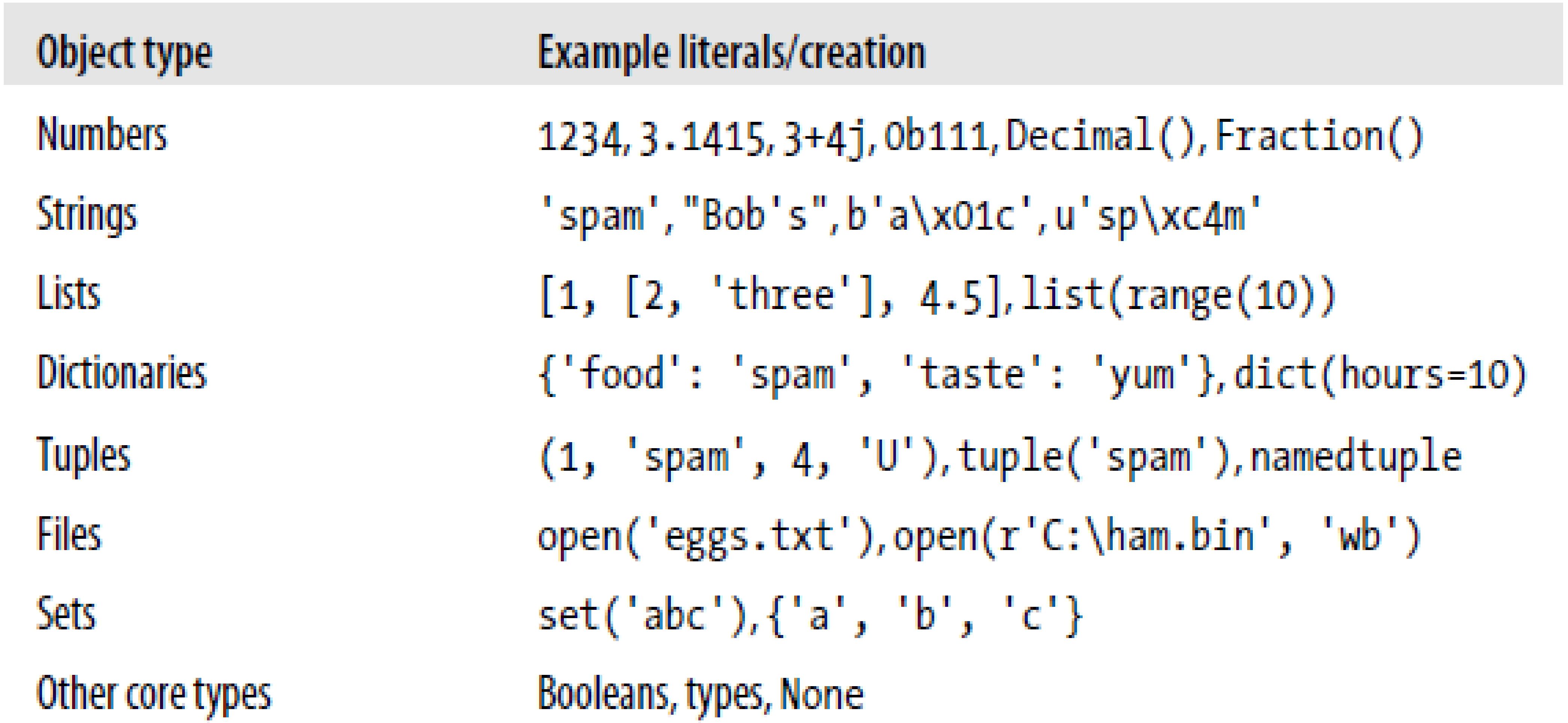
- help("package_name")

Basic operations


Python 3.6 (64-bit)

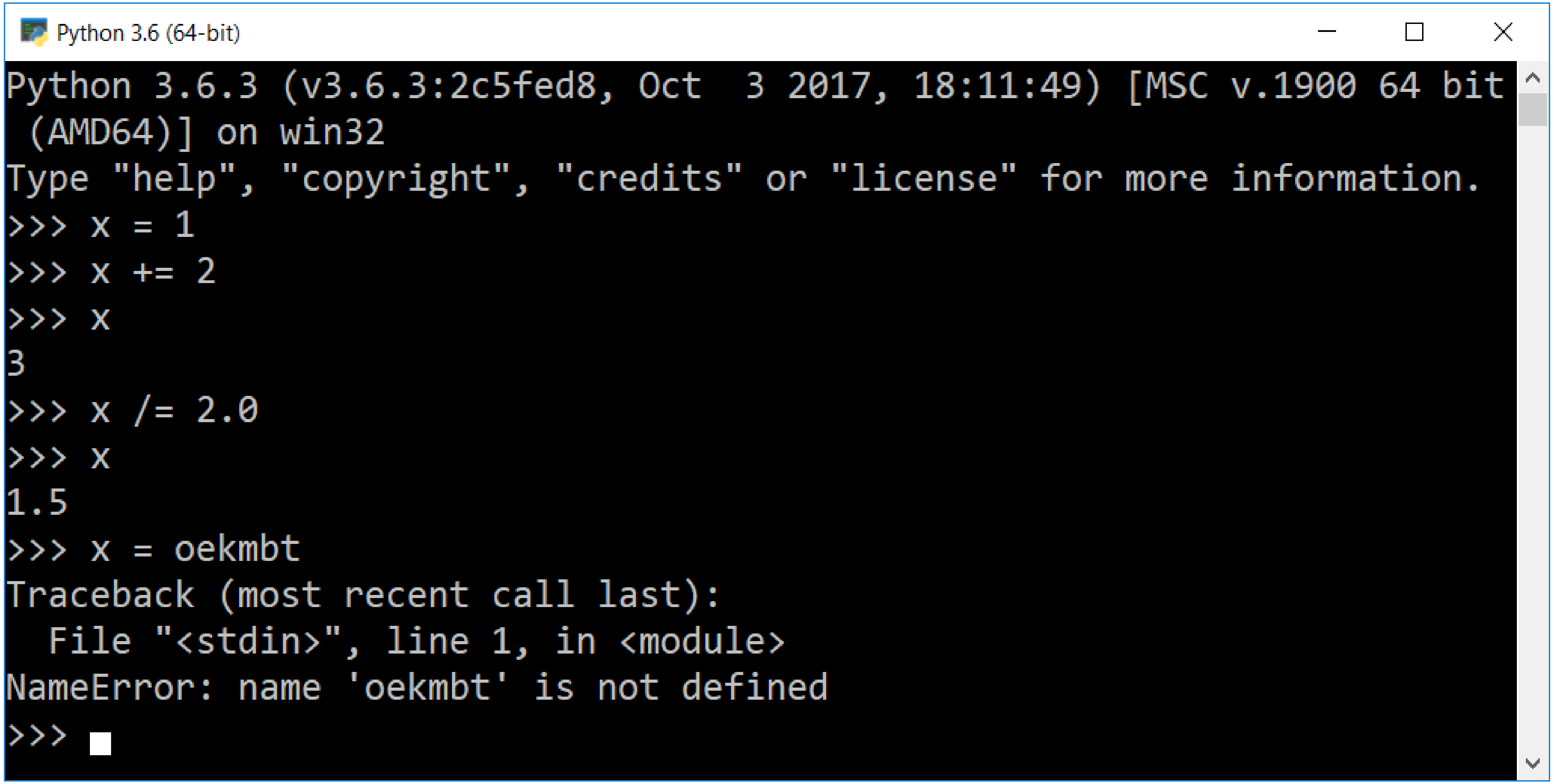
```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> b = 5
>>> a+b
8
>>> 50 - a + b
52
>>> b / a
1.6666666666666667
>>> c = 5.0
>>> c / a
1.6666666666666667
>>> b ** a
125
>>>
```

Core data types



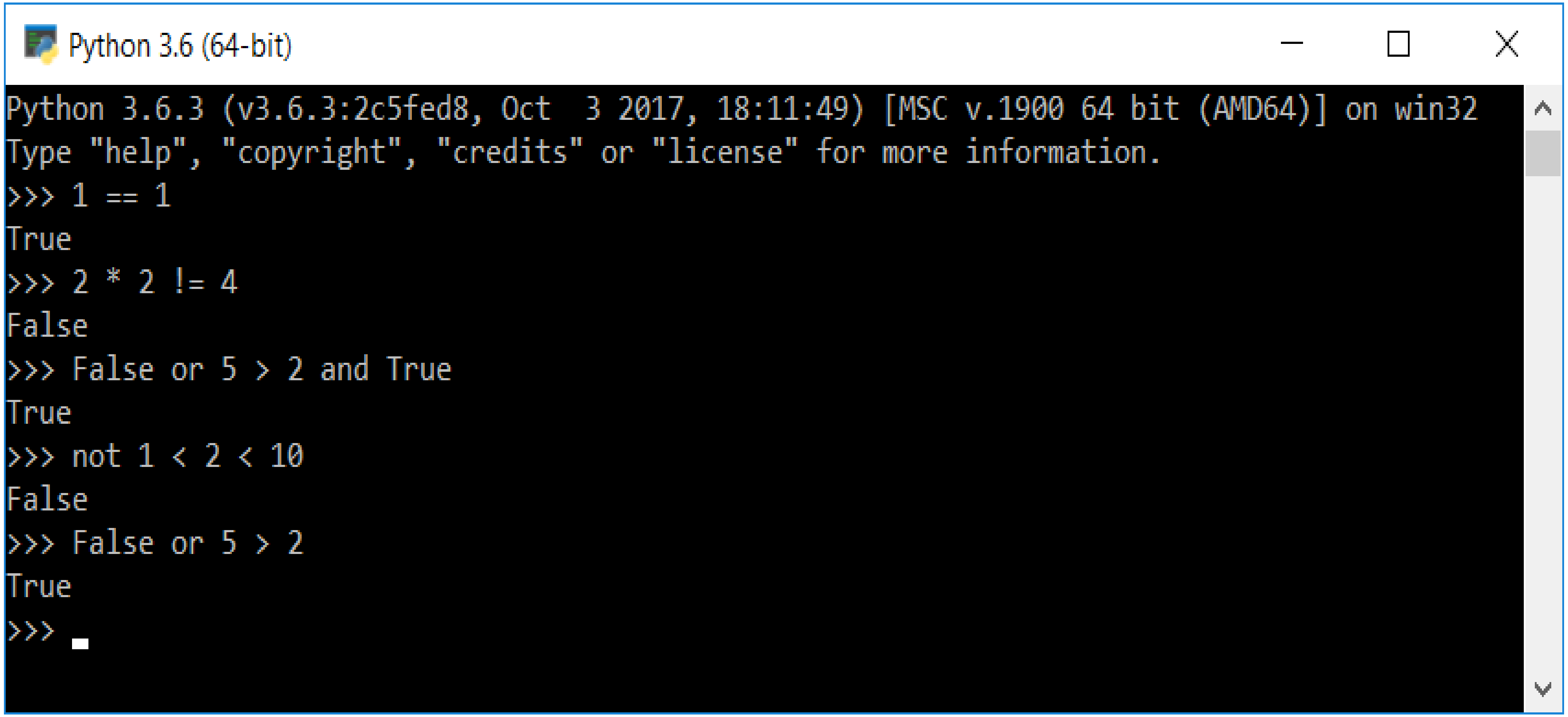
Object type	Example literals/creation
Numbers	`1234`, `3.1415`, `3+4j`, `0b111`, `Decimal()`, `Fraction()`
Strings	`'spam'`, `"Bob's"`, `b'a\x01c'`, `u'sp\xc4m'`
Lists	`[1, [2, 'three'], 4.5]`, `list(range(10))`
Dictionaries	`{'food': 'spam', 'taste': 'yum'}`, `dict(hours=10)`
Tuples	`(1, 'spam', 4, 'U')`, `tuple('spam')`, `namedtuple`
Files	`open('eggs.txt')`, `open(r'C:\ham.bin', 'wb')`
Sets	`set('abc')`, `{'a', 'b', 'c'}`
Other core types	Booleans, types, None

Strongly typed


Python 3.6 (64-bit)

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1
>>> x += 2
>>> x
3
>>> x /= 2.0
>>> x
1.5
>>> x = oekmbt
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'oekmbt' is not defined
>>>
```

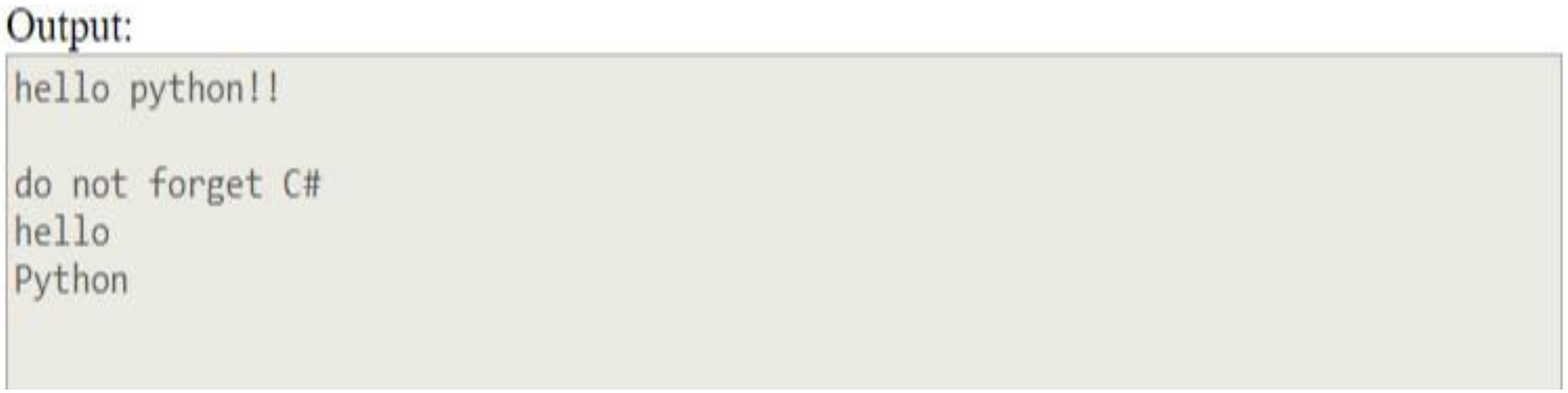
Boolean expression



```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 == 1
True
>>> 2 * 2 != 4
False
>>> False or 5 > 2 and True
True
>>> not 1 < 2 < 10
False
>>> False or 5 > 2
True
>>>
```

String expression

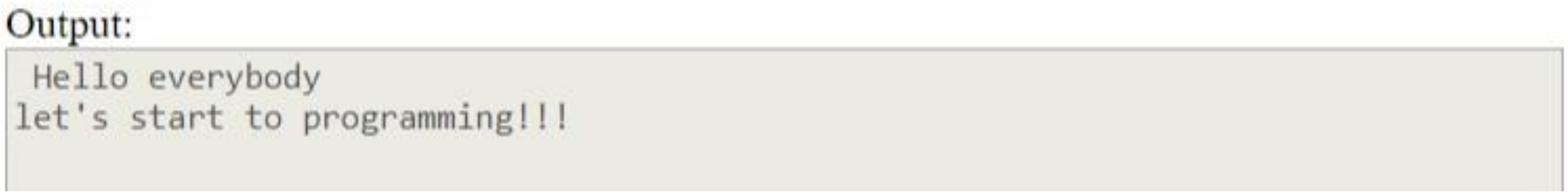
- print("hello python!!")
- print
- print("do not forget C#")
- print("hello\nPython")


Output:

```
hello python!!

do not forget C#
hello
Python
```

- print (""" Hello everybody

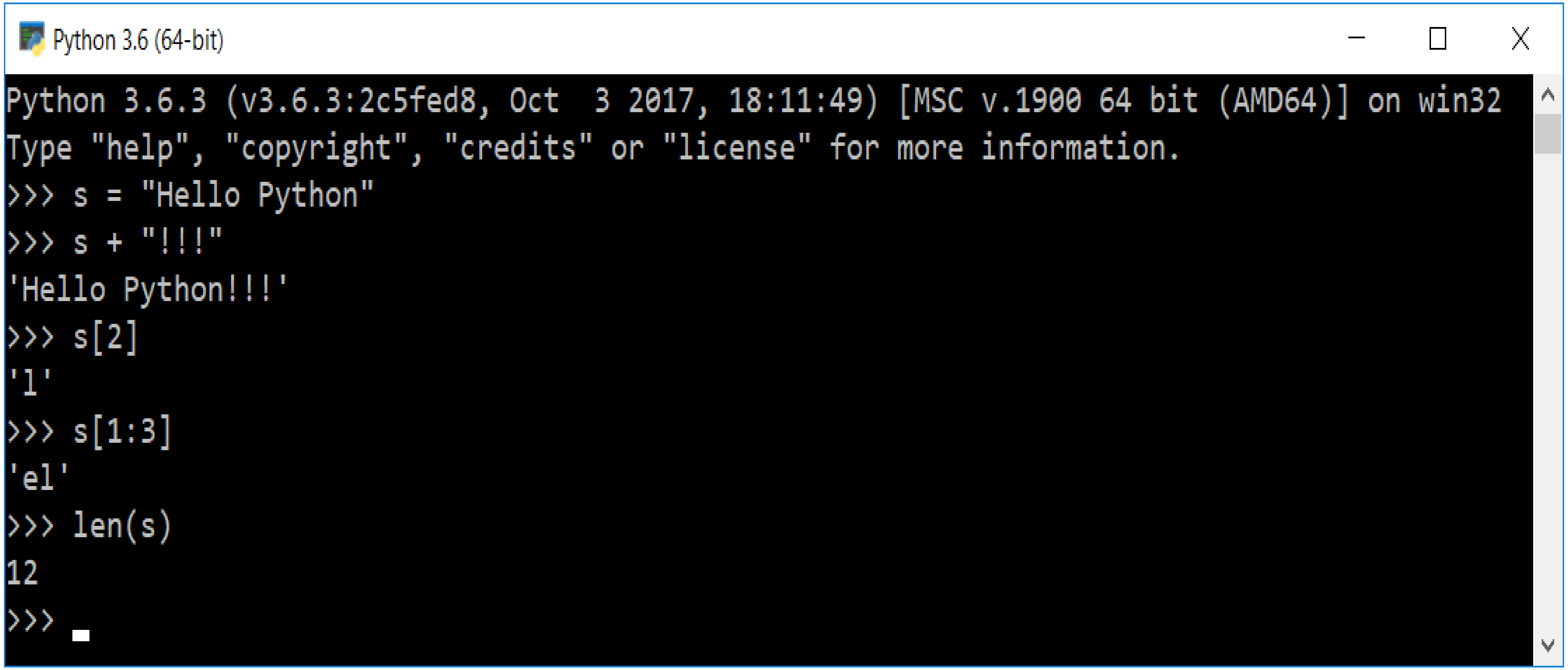
let's start to programming!!!""")



Output:

```
 Hello everybody
let's start to programming!!!
```

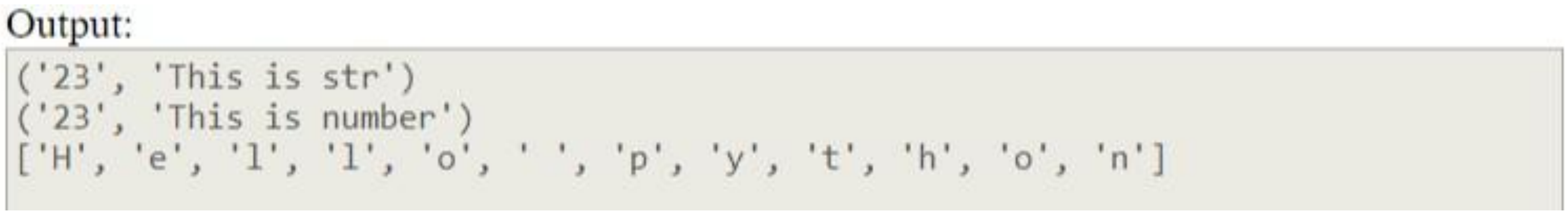
str = string



Python 3.6 (64-bit)

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> s = "Hello Python"
>>> s + "!!!"
'Hello Python!!!'
>>> s[2]
'l'
>>> s[1:3]
'el'
>>> len(s)
12
>>> _
```

- x = "23"
- print(x, "This is str")
- y = int(x)
- print(x, "This is number")
- s = "Hello python"
- arr = list(s)
- print(arr)


Output:

```
('23', 'This is str')
('23', 'This is number')
['H', 'e', 'l', 'l', 'o', ' ', 'p', 'y', 't', 'h', 'o', 'n']
```

Symbol coding

```
# -*- coding: utf-8 -*-

# coding: utf8
```

If

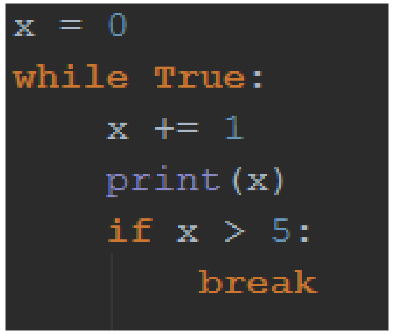
```
a = int(input())
if a < -5:
    print('Low')
elif -5 <= a <= 5:
    print('Mid')
else:
    print('High')
```

While

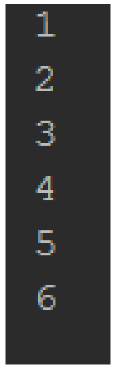
```
p = 4
while p > 0:
    p -= 1
    print(p)
```

?????

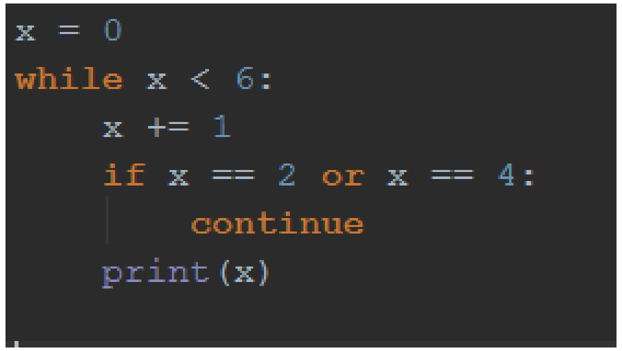
Break/ Continue



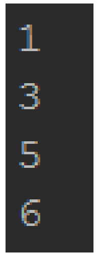
```
x = 0
while True:
    x += 1
    print(x)
    if x > 5:
        break
```



```
1
2
3
4
5
6
```



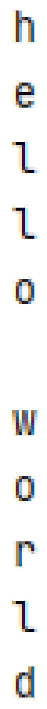
```
x = 0
while x < 6:
    x += 1
    if x == 2 or x == 4:
        continue
    print(x)
```



```
1
3
5
6
```

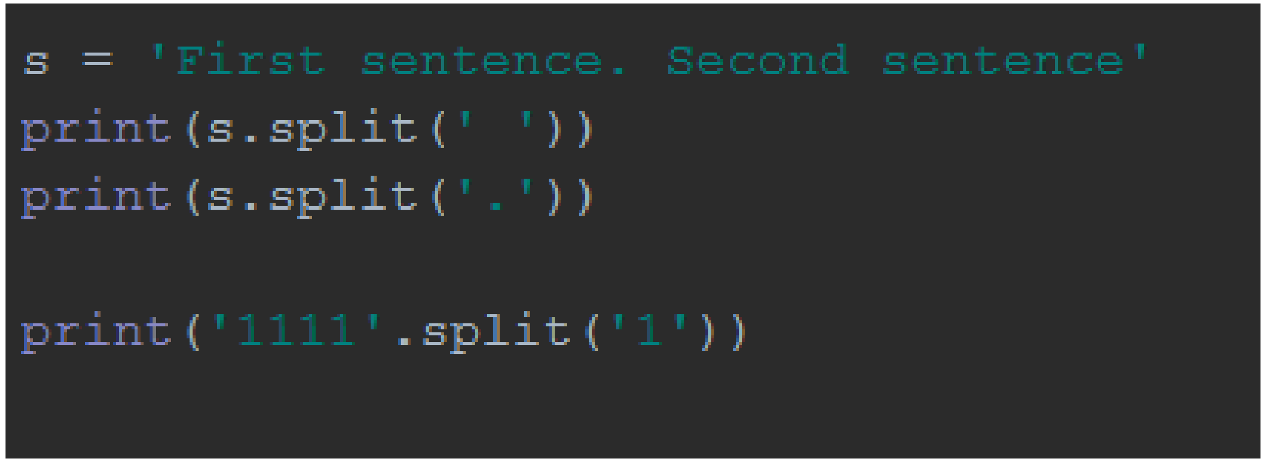
For

```
data = 'hello world‘

for i in data:
    print(i)


for i in range(10):
    i+=1
```

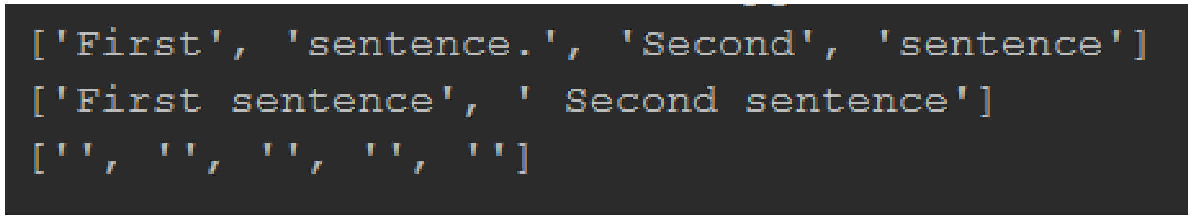


```
h
e
l
l
o
 
w
o
r
l
d
```

Type-Specific Methods. Split for strings

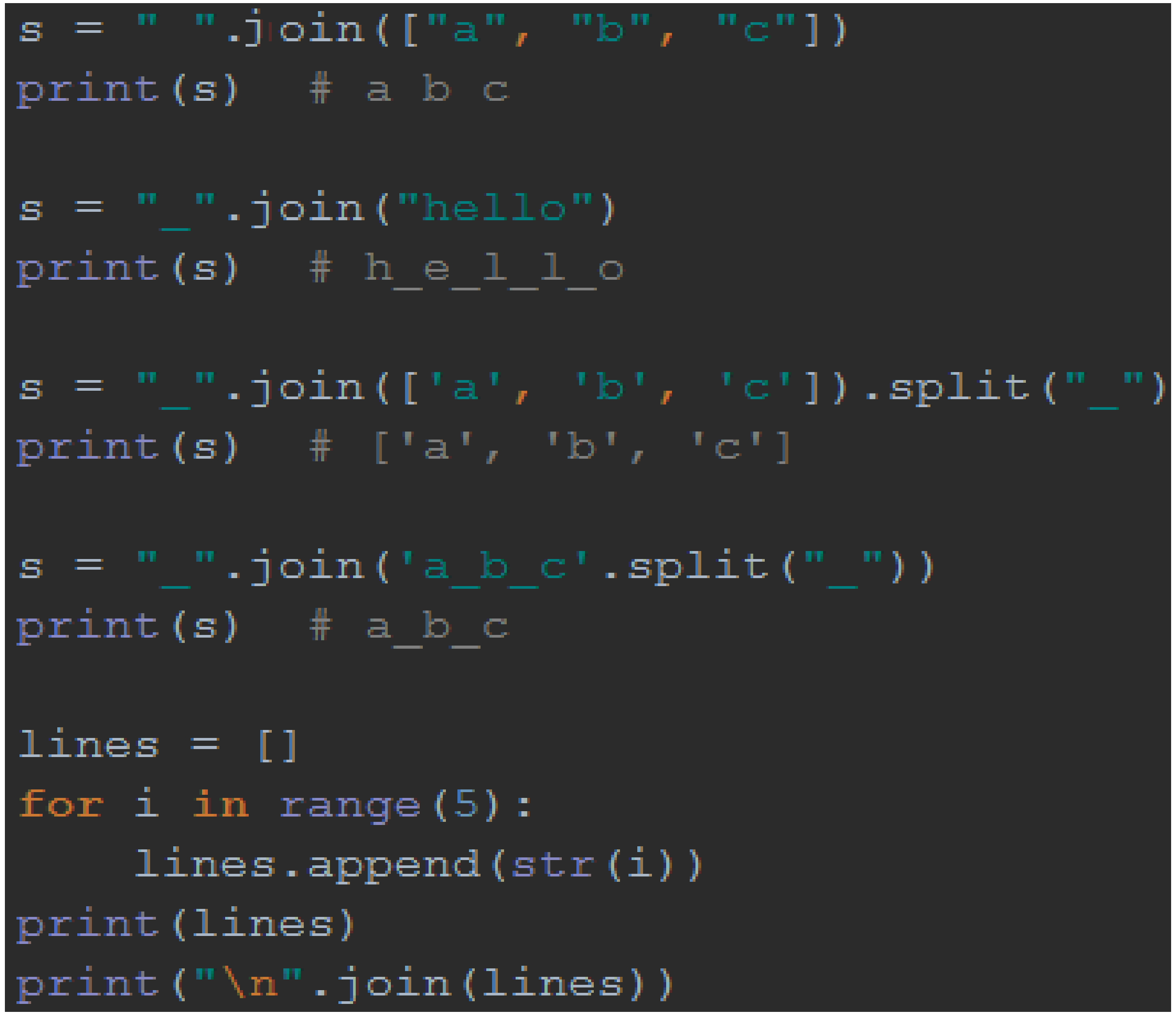


```python
s = 'First sentence. Second sentence'
print(s.split(' '))
print(s.split('.'))

print('1111'.split('1'))
```



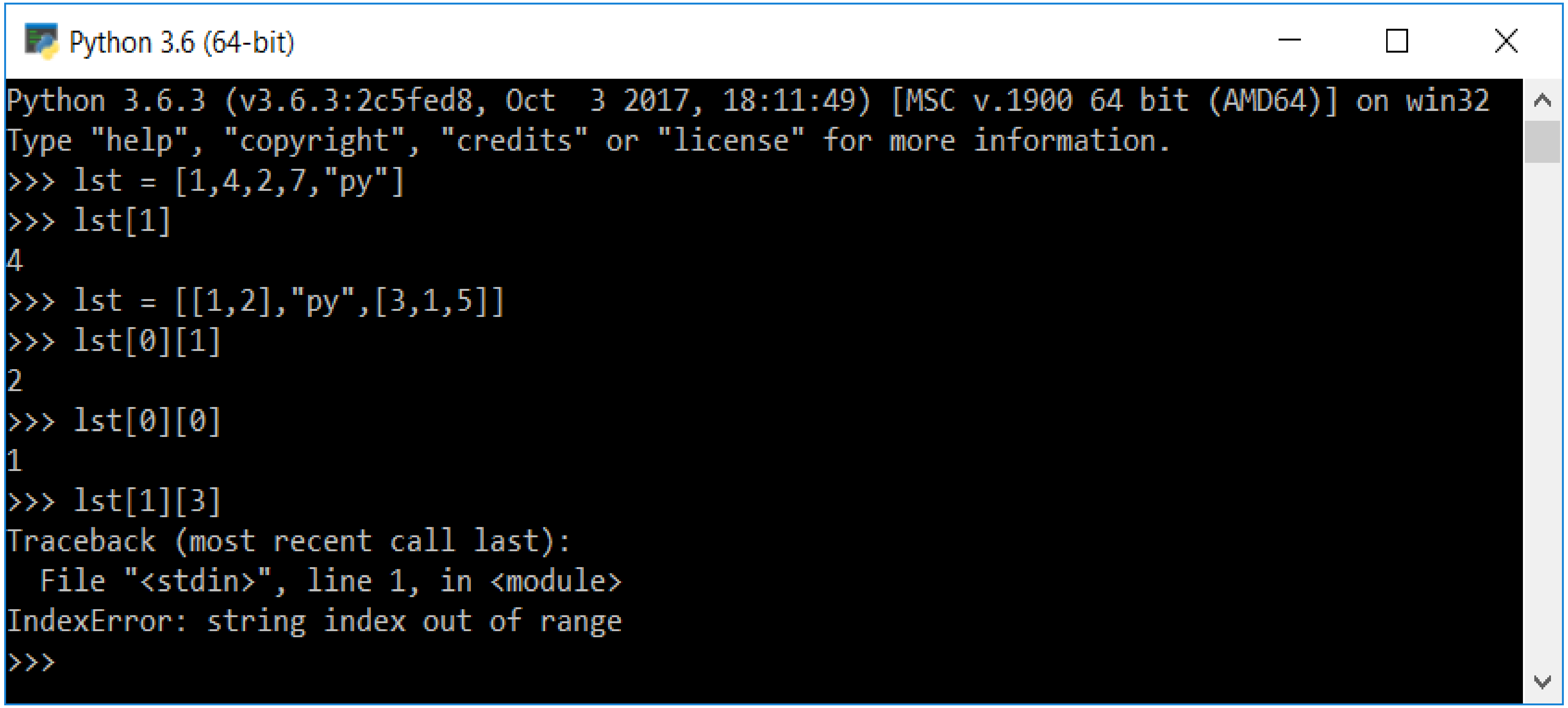
```
['First', 'sentence.', 'Second', 'sentence']
['First sentence', ' Second sentence']
['', '', '', '', '']
```

Type-Specific Methods. Join for strings



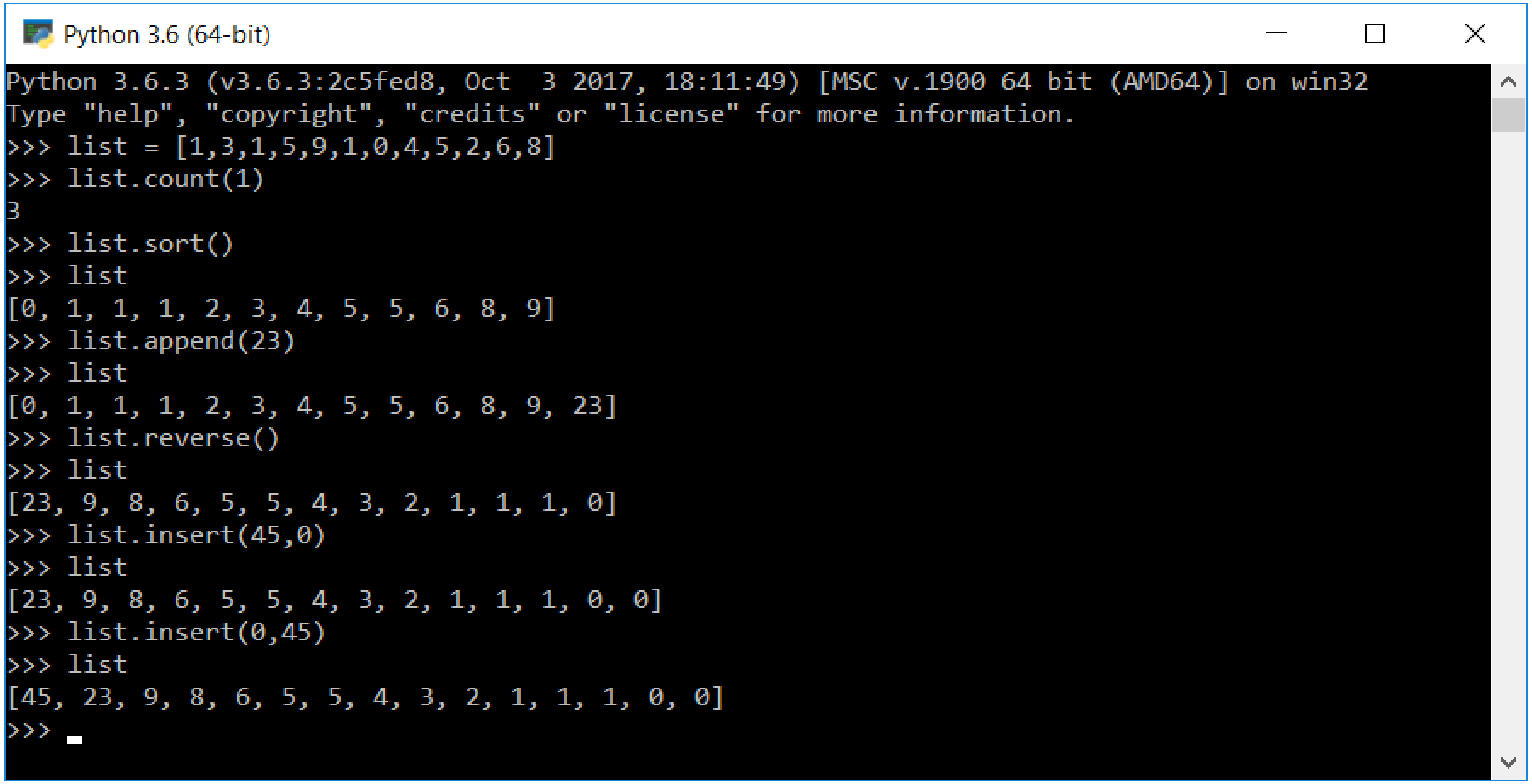
```
s = " ".join(["a", "b", "c"])
print(s)  # a b c

s = "_".join("hello")
print(s)  # h_e_l_l_o

s = "_".join(['a', 'b', 'c']).split("_")
print(s)  # ['a', 'b', 'c']

s = "_".join('a_b_c'.split("_"))
print(s)  # a_b_c

lines = []
for i in range(5):
    lines.append(str(i))
print(lines)
print("\n".join(lines))
```

List. Index


Python 3.6 (64-bit)

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> lst = [1,4,2,7,"py"]
>>> lst[1]
4
>>> lst = [[1,2],"py",[3,1,5]]
>>> lst[0][1]
2
>>> lst[0][0]
1
>>> lst[1][3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
```

List

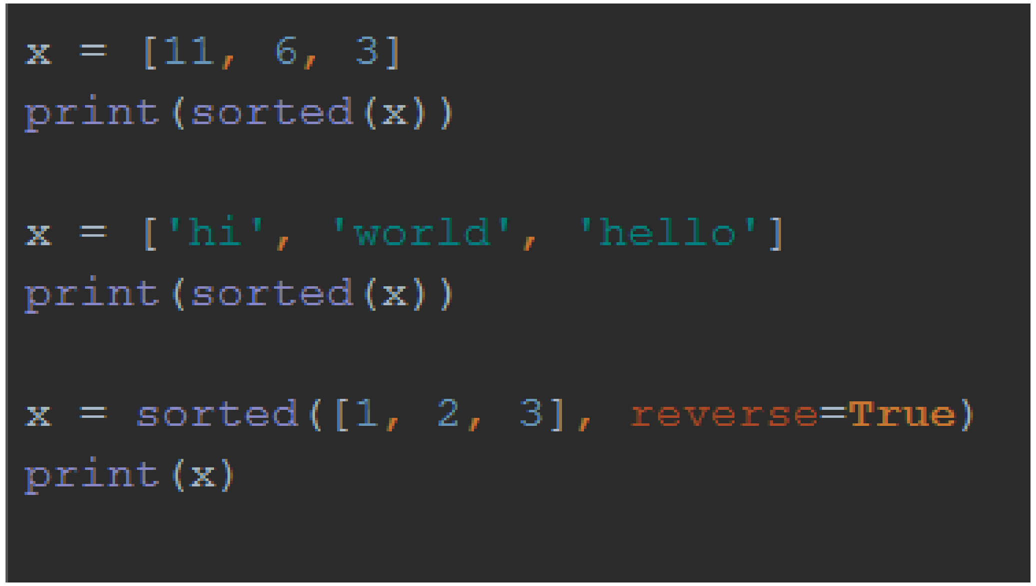


```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> list = [1,3,1,5,9,1,0,4,5,2,6,8]
>>> list.count(1)
3
>>> list.sort()
>>> list
[0, 1, 1, 1, 2, 3, 4, 5, 5, 6, 8, 9]
>>> list.append(23)
>>> list
[0, 1, 1, 1, 2, 3, 4, 5, 5, 6, 8, 9, 23]
>>> list.reverse()
>>> list
[23, 9, 8, 6, 5, 5, 4, 3, 2, 1, 1, 1, 0]
>>> list.insert(45,0)
>>> list
[23, 9, 8, 6, 5, 5, 4, 3, 2, 1, 1, 1, 0, 0]
>>> list.insert(0,45)
>>> list
[45, 23, 9, 8, 6, 5, 5, 4, 3, 2, 1, 1, 1, 0, 0]
>>>
```

List methods

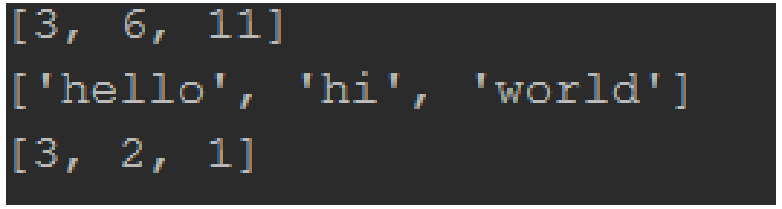
- list.append(x)
- list.extend(lst)
- list.insert(i,x)
- list.remove(x)
- list.index(x,[start[,end]])
- list.count(x)
- list.reverse()
- list.clear()
- etc python.org

Sorted for list



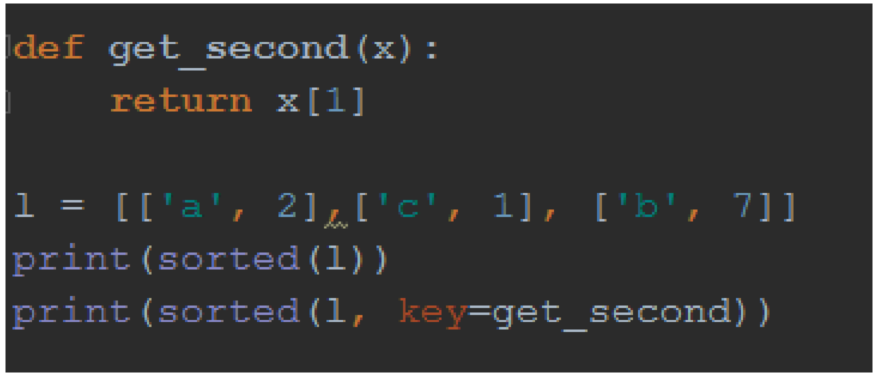
```
x = [11, 6, 3]
print(sorted(x))

x = ['hi', 'world', 'hello']
print(sorted(x))

x = sorted([1, 2, 3], reverse=True)
print(x)
```

Output:



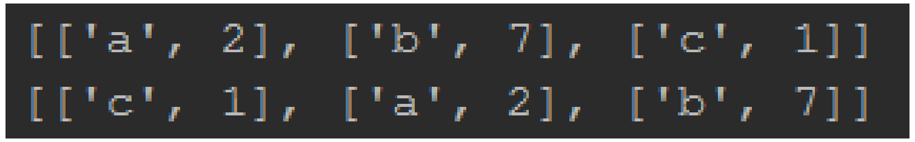
```
[3, 6, 11]
['hello', 'hi', 'world']
[3, 2, 1]
```

Sorted options



```python
def get_second(x):
    return x[1]

l = [['a', 2],['c', 1], ['b', 7]]
print(sorted(l))
print(sorted(l, key=get_second))
```

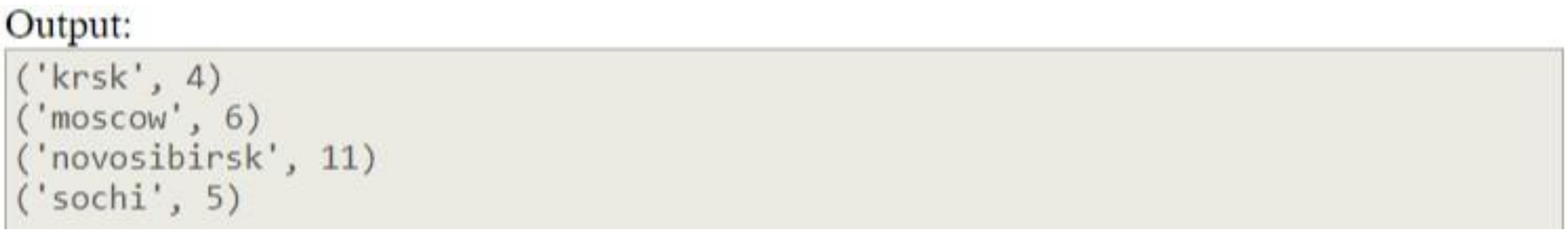
Output:



```
[['a', 2], ['b', 7], ['c', 1]]
[['c', 1], ['a', 2], ['b', 7]]
```

For

```
lst = ["krsk","moscow","novosibirsk", "sochi"]
for x in lst:
    print(x, len(x))
```

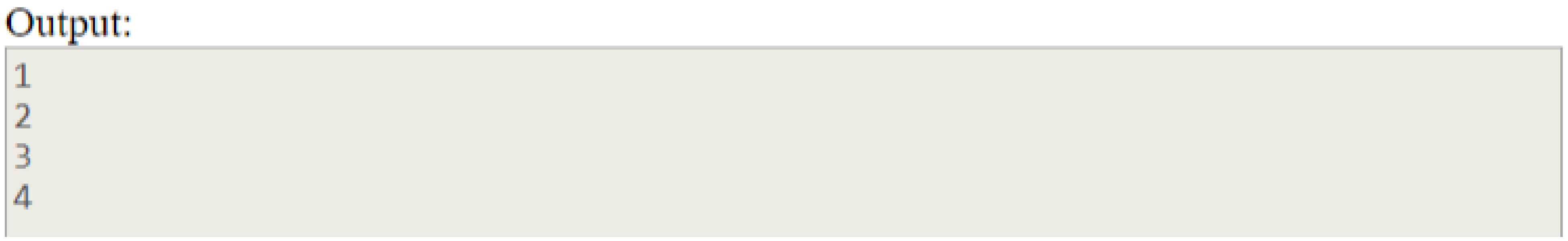


Output:

```
('krsk', 4)
('moscow', 6)
('novosibirsk', 11)
('sochi', 5)
```

For

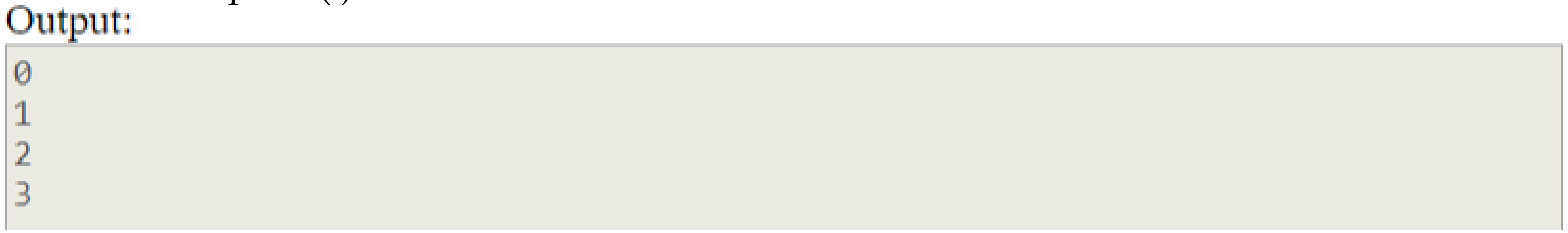
```
for i in range(1,5):
    print(i)
```



Output:

```
1
2
3
4
```

```
lst = ["krsk","moscow","novosibirsk", "sochi"]
for i in range(len(lst)):
    print(i)
```



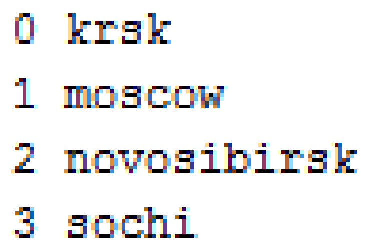
Output:

```
0
1
2
3
```

For / enumerate

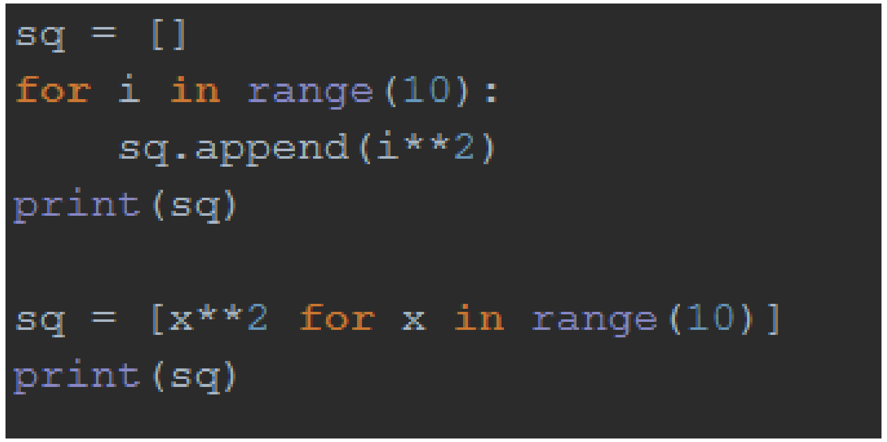
```
lst = ["krsk","moscow","novosibirsk", "sochi"]
for num, el in enumerate(lst):
    print(num, el)
```

Output:



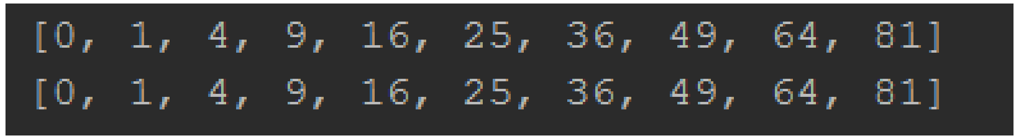
```
0 krsk
1 moscow
2 novosibirsk
3 sochi
```

Type-Specific Methods. **List comprehensions**



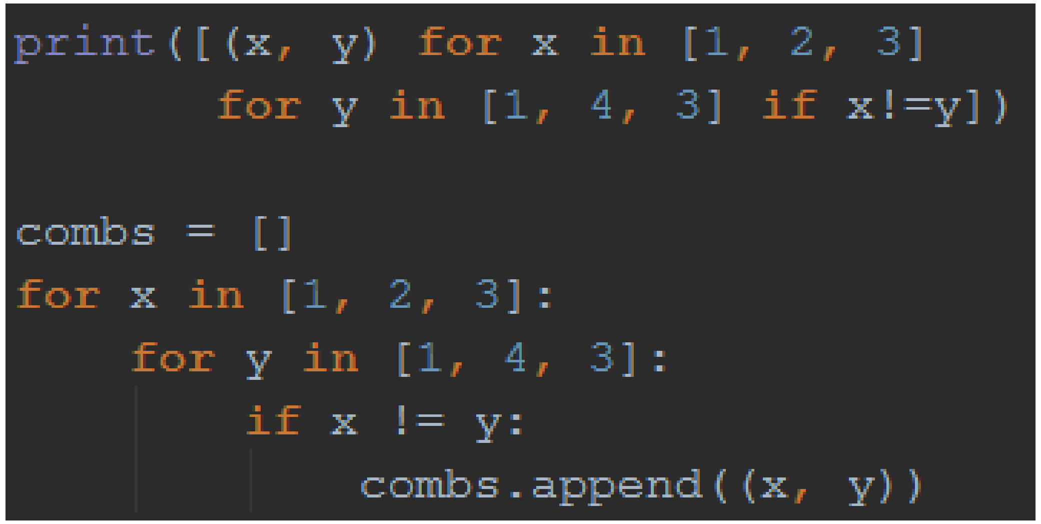
```
sq = []
for i in range(10):
    sq.append(i**2)
print(sq)

sq = [x**2 for x in range(10)]
print(sq)
```

Output:



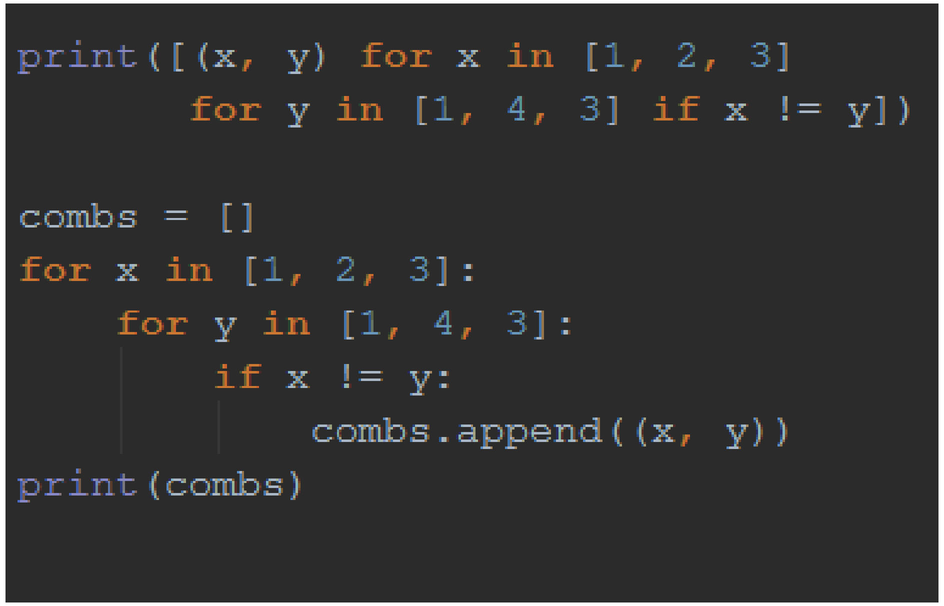
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Complex list comprehensions



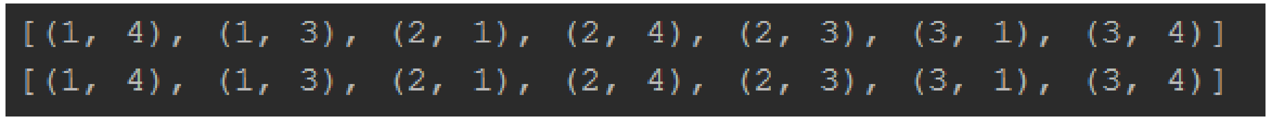
```python
print([(x, y) for x in [1, 2, 3]
        for y in [1, 4, 3] if x!=y])

combs = []
for x in [1, 2, 3]:
    for y in [1, 4, 3]:
        if x != y:
            combs.append((x, y))
```

Complex list expressions



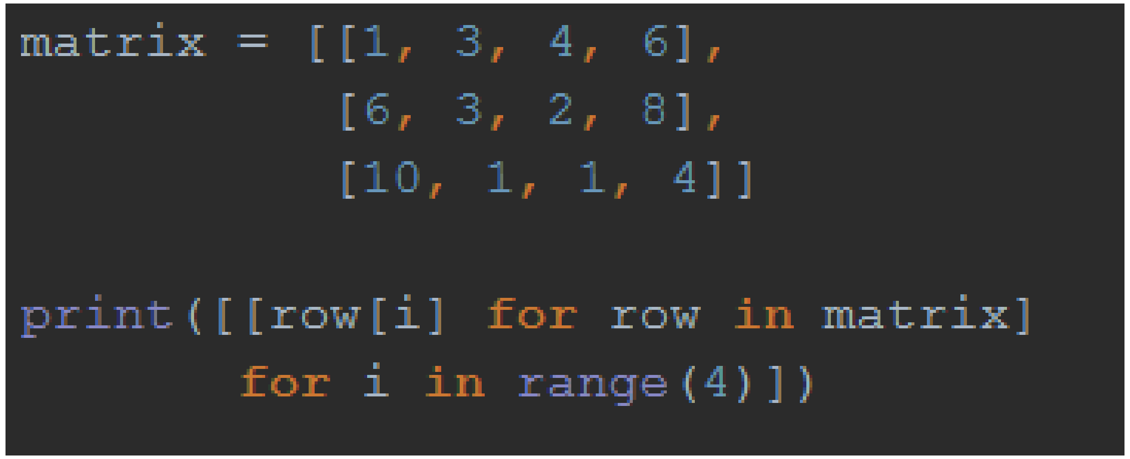
```python
print([(x, y) for x in [1, 2, 3]
       for y in [1, 4, 3] if x != y])

combs = []
for x in [1, 2, 3]:
    for y in [1, 4, 3]:
        if x != y:
            combs.append((x, y))
print(combs)
```

Output:



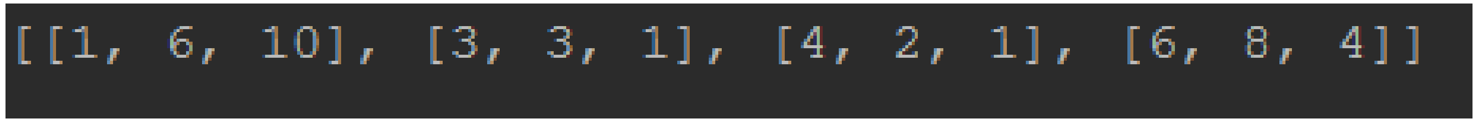
```
[(1, 4), (1, 3), (2, 1), (2, 4), (2, 3), (3, 1), (3, 4)]
[(1, 4), (1, 3), (2, 1), (2, 4), (2, 3), (3, 1), (3, 4)]
```

Complex list comprehensions



```
matrix = [[1, 3, 4, 6],
          [6, 3, 2, 8],
          [10, 1, 1, 4]]

print([[row[i] for row in matrix]
       for i in range(4)])
```

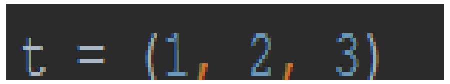
Output:



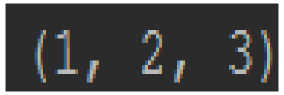
```
[[1, 6, 10], [3, 3, 1], [4, 2, 1], [6, 8, 4]]
```

Tuples

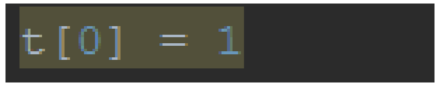
Tuples are sequences, like lists, but they are immutable, like strings and cannot be changed



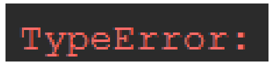
```
t = (1, 2, 3)
```



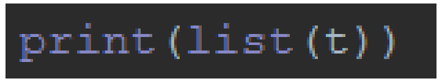
```
(1, 2, 3)
```



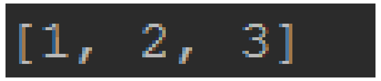
```
t[0] = 1
```



```
TypeError:
```

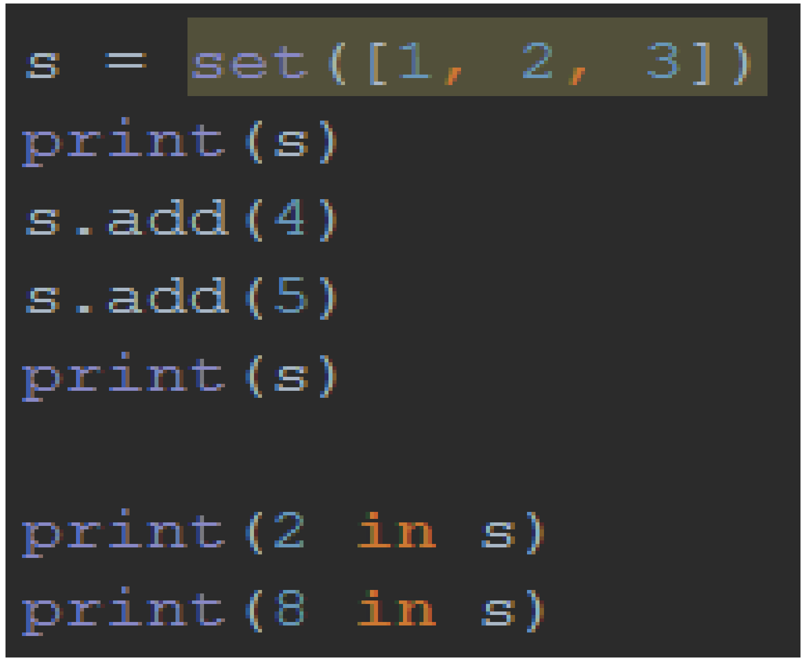


```
print(list(t))
```



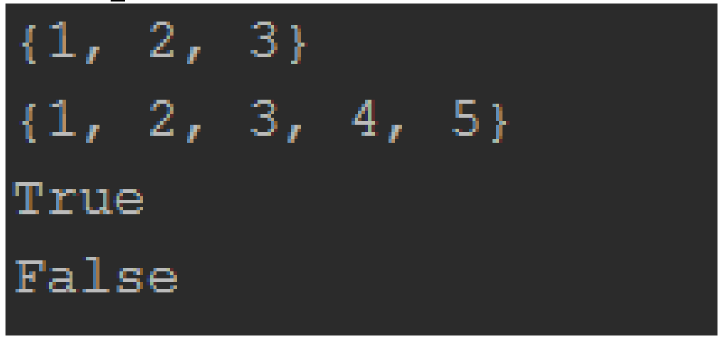
```
[1, 2, 3]
```

Set



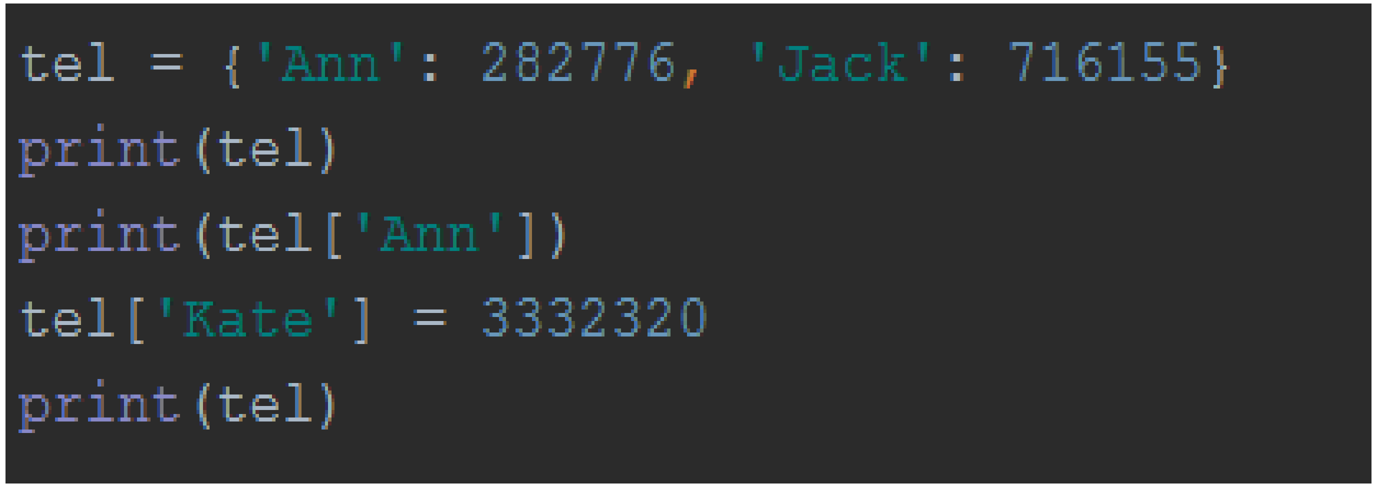
```
s = set([1, 2, 3])
print(s)
s.add(4)
s.add(5)
print(s)

print(2 in s)
print(8 in s)
```

Output:

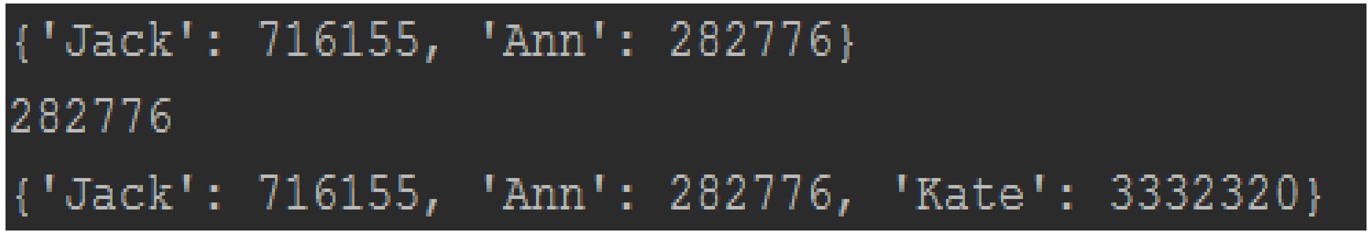


```
{1, 2, 3}
{1, 2, 3, 4, 5}
True
False
```

Dictionary

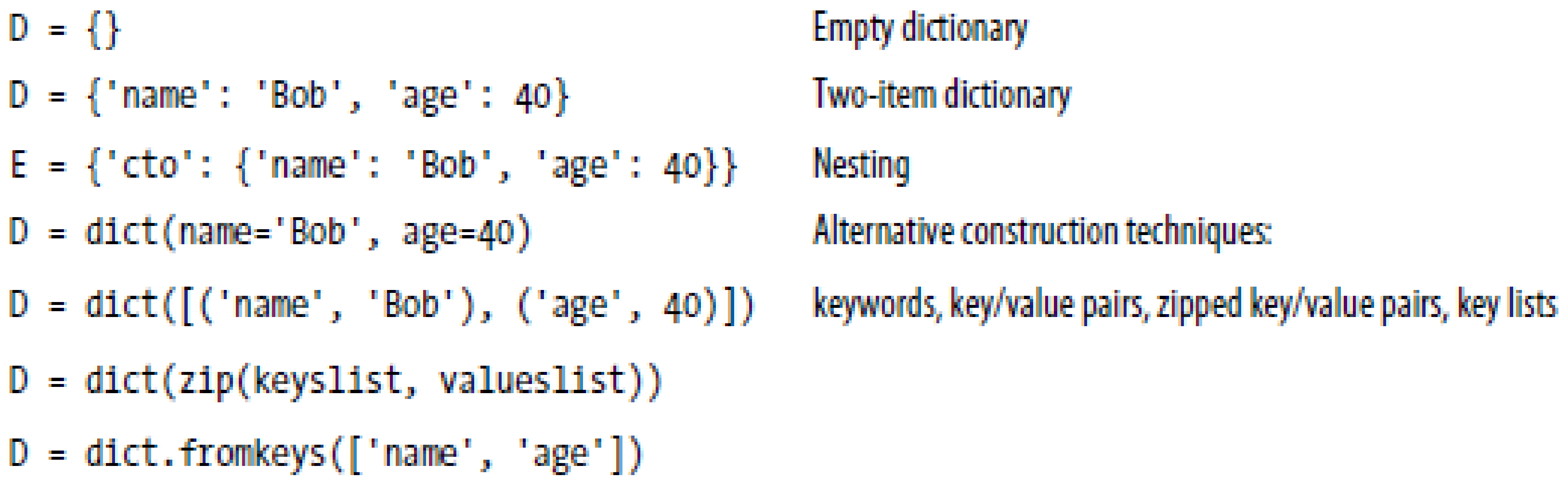


```python
tel = {'Ann': 282776, 'Jack': 716155}
print(tel)
print(tel['Ann'])
tel['Kate'] = 3332320
print(tel)
```



```
{'Jack': 716155, 'Ann': 282776}
282776
{'Jack': 716155, 'Ann': 282776, 'Kate': 3332320}
```

Create Dictionary



`D = {}`	Empty dictionary
`D = {'name': 'Bob', 'age': 40}`	Two-item dictionary
`E = {'cto': {'name': 'Bob', 'age': 40}}`	Nesting
`D = dict(name='Bob', age=40)`	Alternative construction techniques:
`D = dict([('name', 'Bob'), ('age', 40)])`	keywords, key/value pairs, zipped key/value pairs, key lists
`D = dict(zip(keyslist, valueslist))`	
`D = dict.fromkeys(['name', 'age'])`	

Keys

```
d = {}

d['a'] = 1
d[5] = 'c'
print(d)
```

```
{'a': 1, 5: 'c'}
```

```
d[(1, 2)] = 0
```

```
{(1, 2): 0}
```

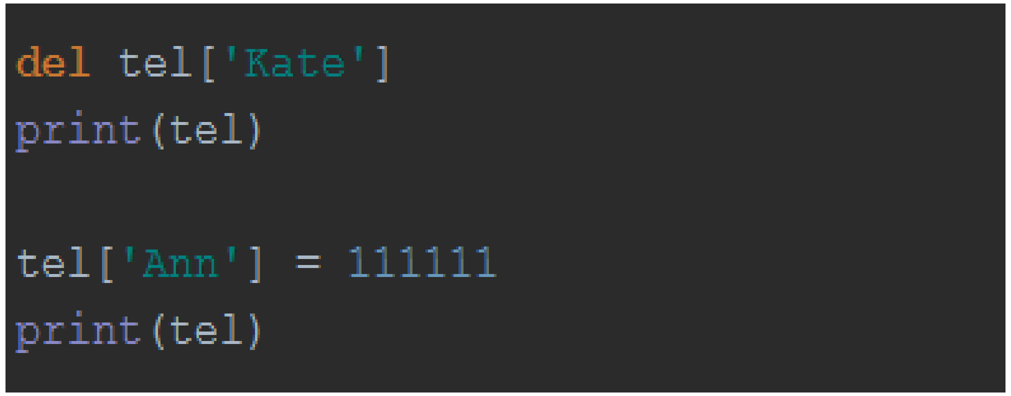
```
d[[1, 2]] = 0
```

```
TypeError: unhashable type: 'list'
```

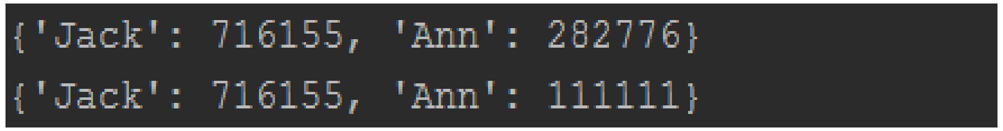
```
d[{'a': 'b'}] = 1
```

```
TypeError: unhashable type: 'dict'
```

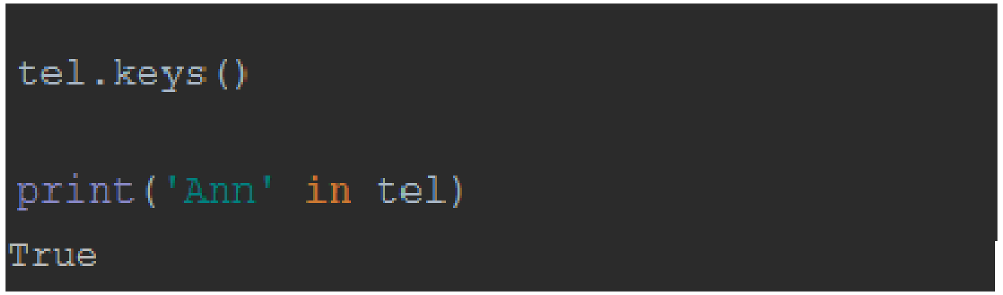
Dictionary operations



```
del tel['Kate']
print(tel)

tel['Ann'] = 111111
print(tel)
```

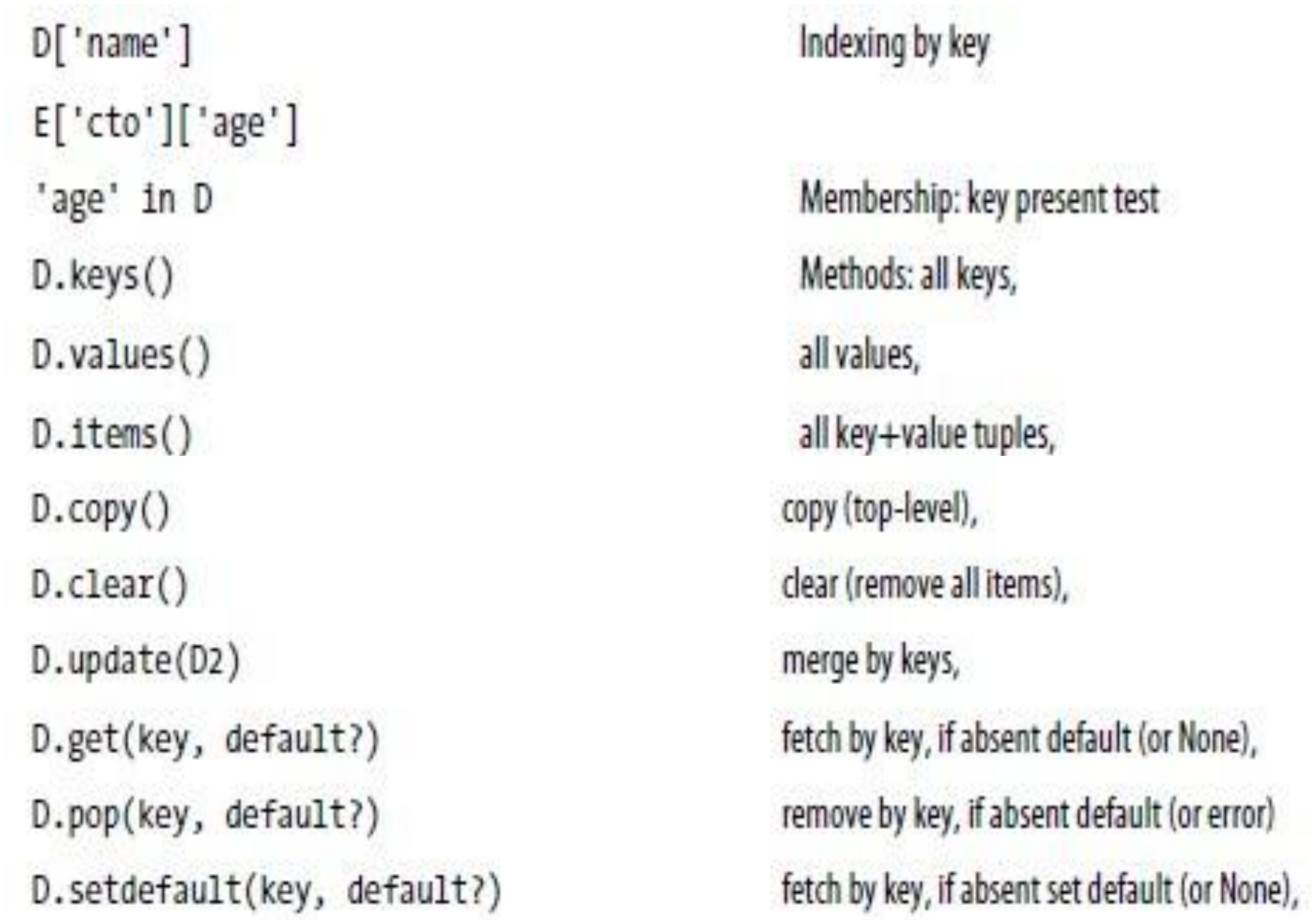


```
{'Jack': 716155, 'Ann': 282776}
{'Jack': 716155, 'Ann': 111111}
```



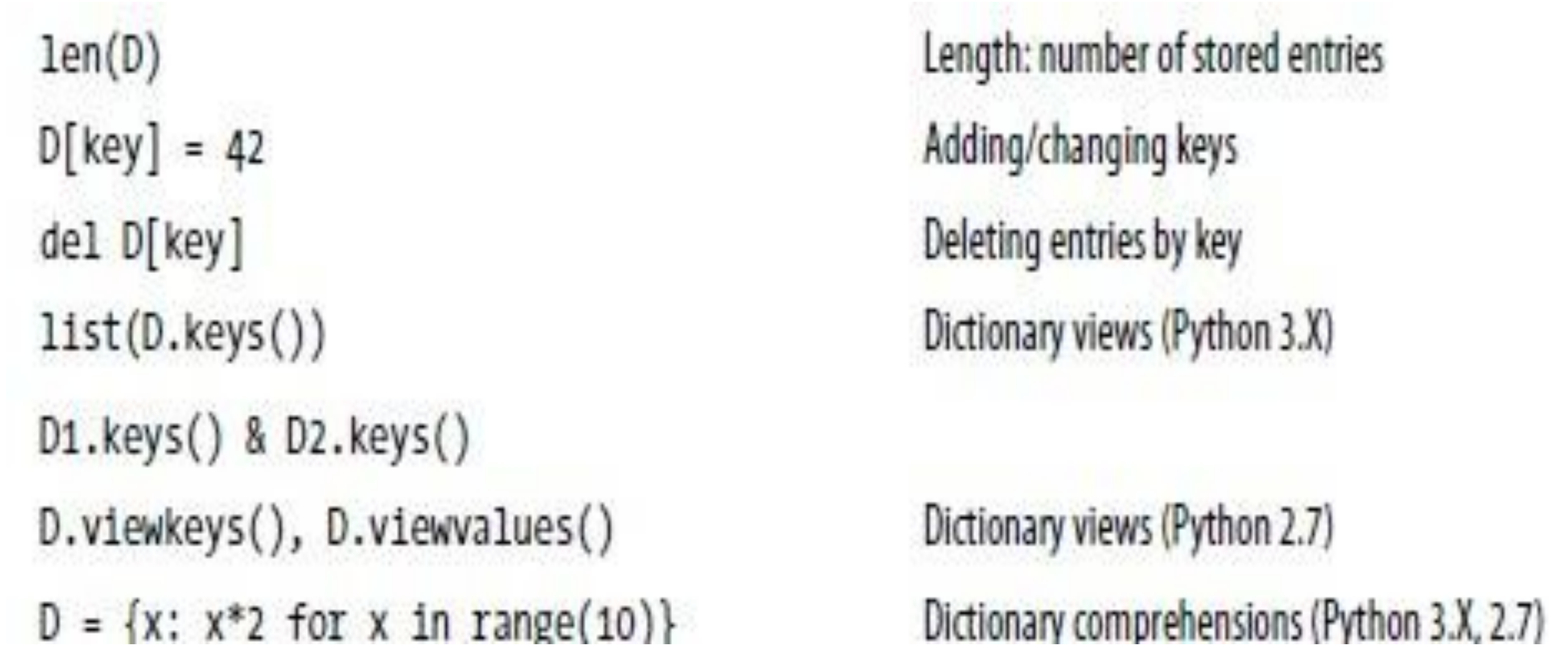
```
tel.keys()

print('Ann' in tel)
True
```

Dictionary operations



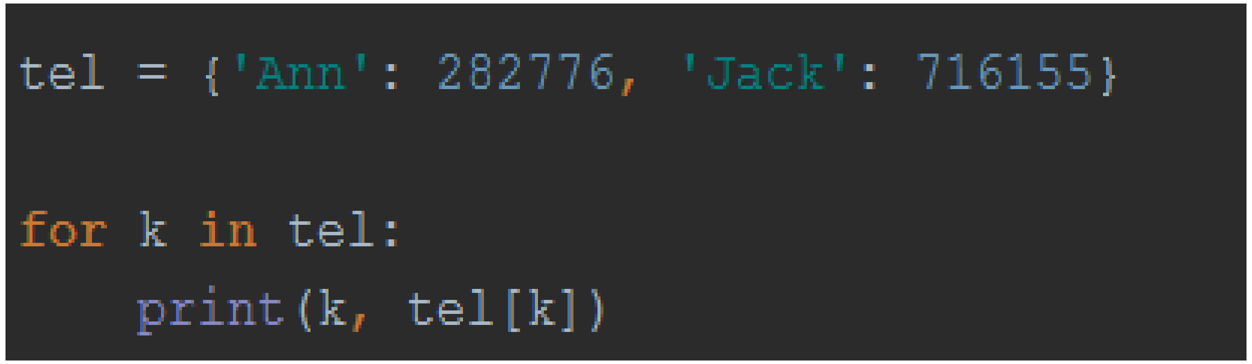
`D['name']`	Indexing by key
`E['cto']['age']`	
`'age' in D`	Membership: key present test
`D.keys()`	Methods: all keys,
`D.values()`	all values,
`D.items()`	all key+value tuples,
`D.copy()`	copy (top-level),
`D.clear()`	clear (remove all items),
`D.update(D2)`	merge by keys,
`D.get(key, default?)`	fetch by key, if absent default (or None),
`D.pop(key, default?)`	remove by key, if absent default (or error)
`D.setdefault(key, default?)`	fetch by key, if absent set default (or None),

Dictionary operations



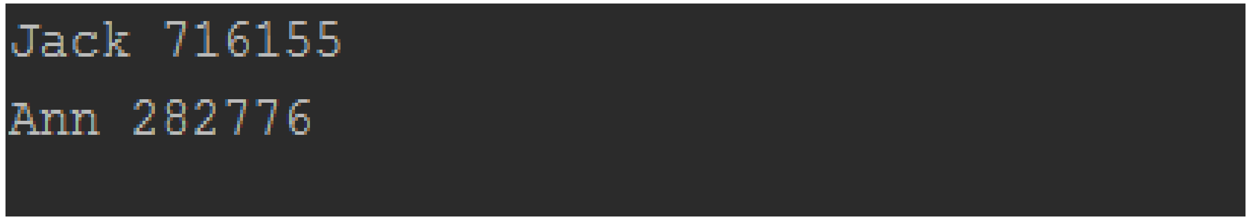
`len(D)`	Length: number of stored entries
`D[key] = 42`	Adding/changing keys
`del D[key]`	Deleting entries by key
`list(D.keys())`	Dictionary views (Python 3.X)
`D1.keys() & D2.keys()`	
`D.viewkeys(), D.viewvalues()`	Dictionary views (Python 2.7)
`D = {x: x*2 for x in range(10)}`	Dictionary comprehensions (Python 3.X, 2.7)

Type-Specific Methods. Dictionary loops



```python
tel = {'Ann': 282776, 'Jack': 716155}

for k in tel:
    print(k, tel[k])
```

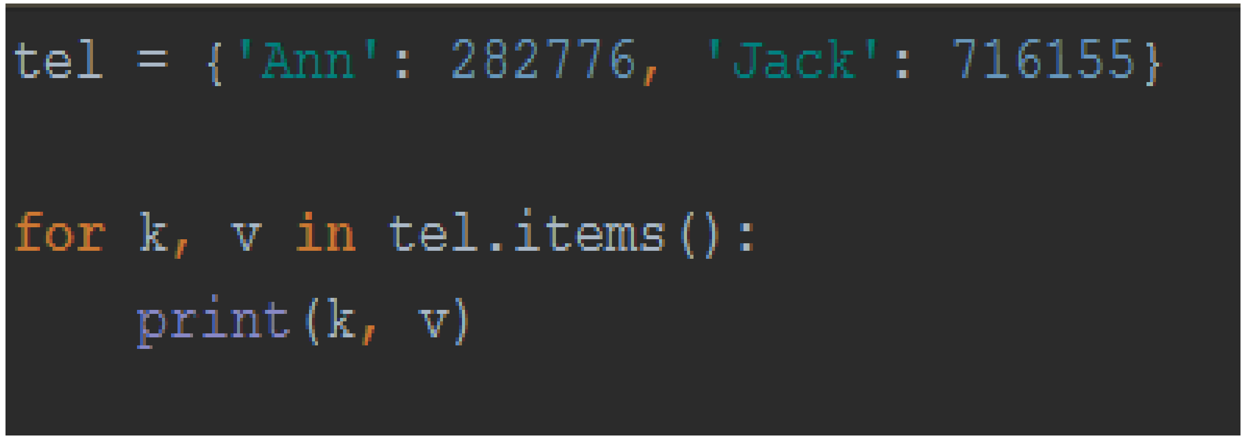
Output:



```
Jack 716155
Ann 282776
```

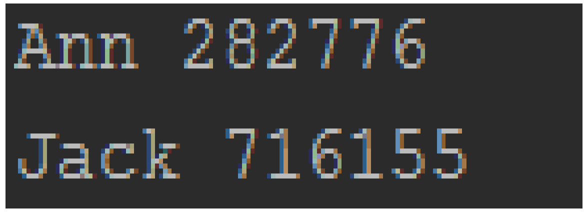
(Random output order)

Dictionary elements



```
tel = {'Ann': 282776, 'Jack': 716155}

for k, v in tel.items():
    print(k, v)
```

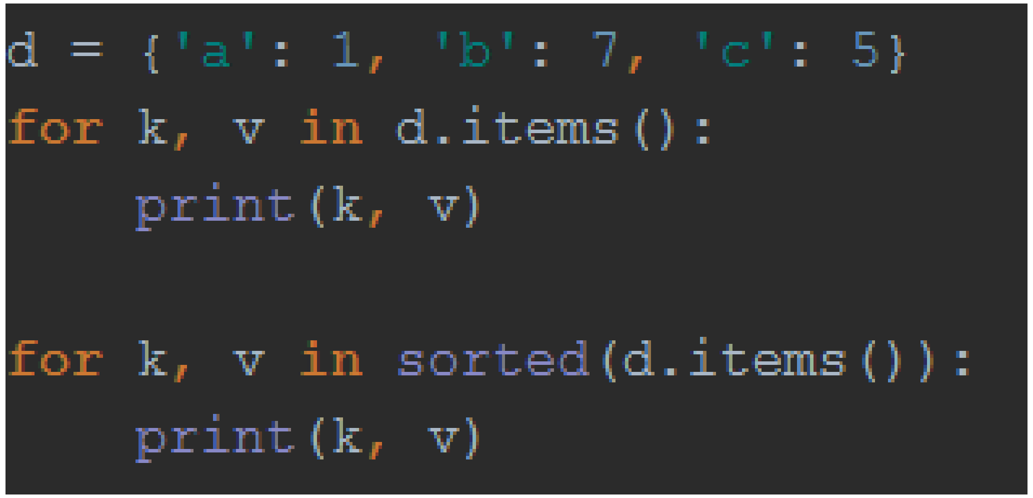
Output:



```
Ann 282776
Jack 716155
```

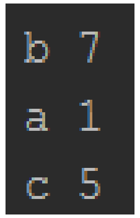
(Random output order)

Ordered dictionary

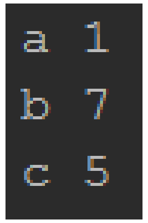


```python
d = {'a': 1, 'b': 7, 'c': 5}
for k, v in d.items():
    print(k, v)

for k, v in sorted(d.items()):
    print(k, v)
```

Output:



```
b 7
a 1
c 5
```



```
a 1
b 7
c 5
```

JSON : JavaScript Object Notation

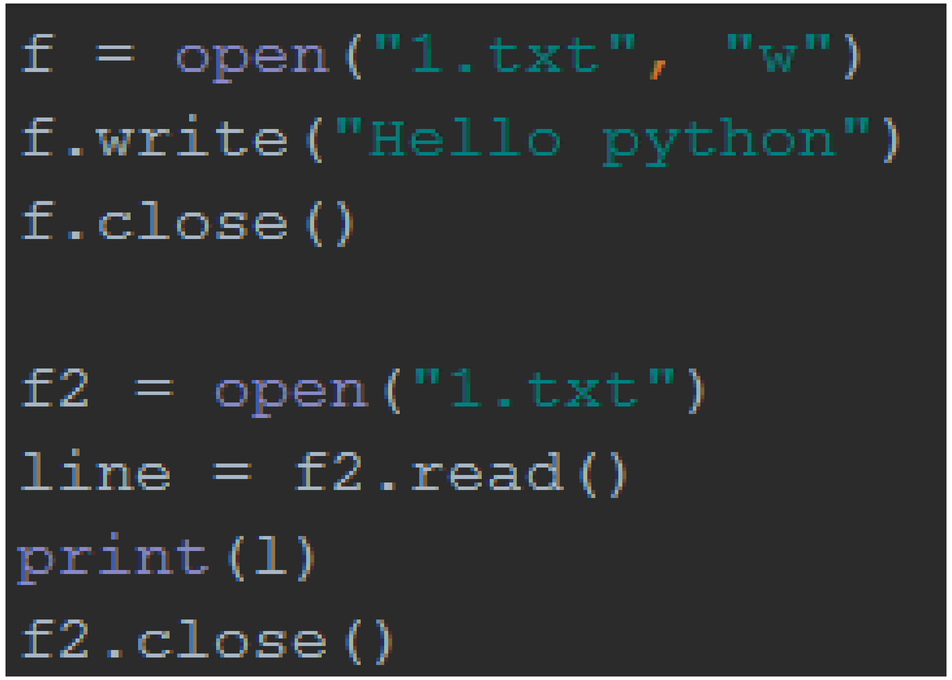
```
a = {
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

JSON

```
import json

with open("data_file.json", "w") as write_file:
    json.dump(a, write_file)

with open("data_file.json", "r") as read_file:
    data = json.load(read_file)
```

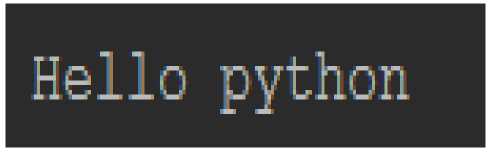
https://python-scripts.com/json

File



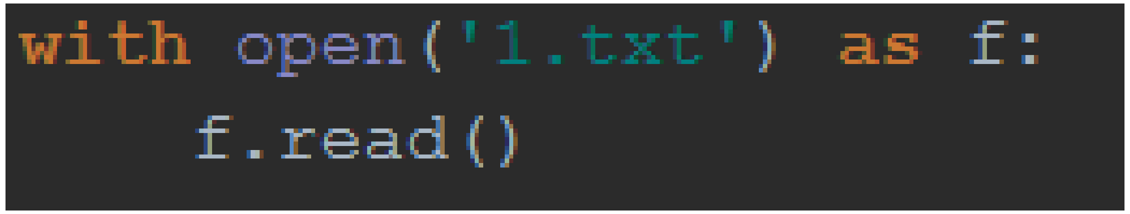
```
f = open("1.txt", "w")
f.write("Hello python")
f.close()

f2 = open("1.txt")
line = f2.read()
print(l)
f2.close()
```

Output:



```
Hello python
```

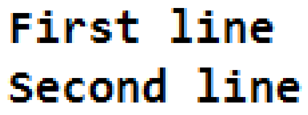
With



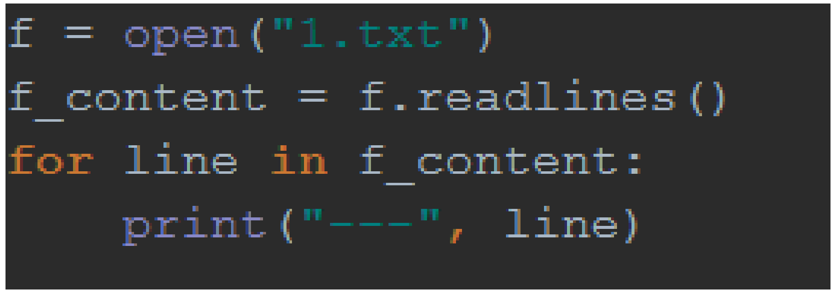
```
with open('1.txt') as f:
    f.read()
```

Reading files

File content:

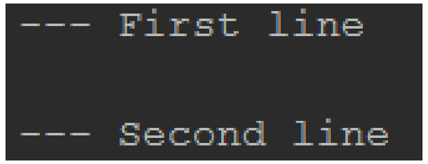


```
First line
Second line
```



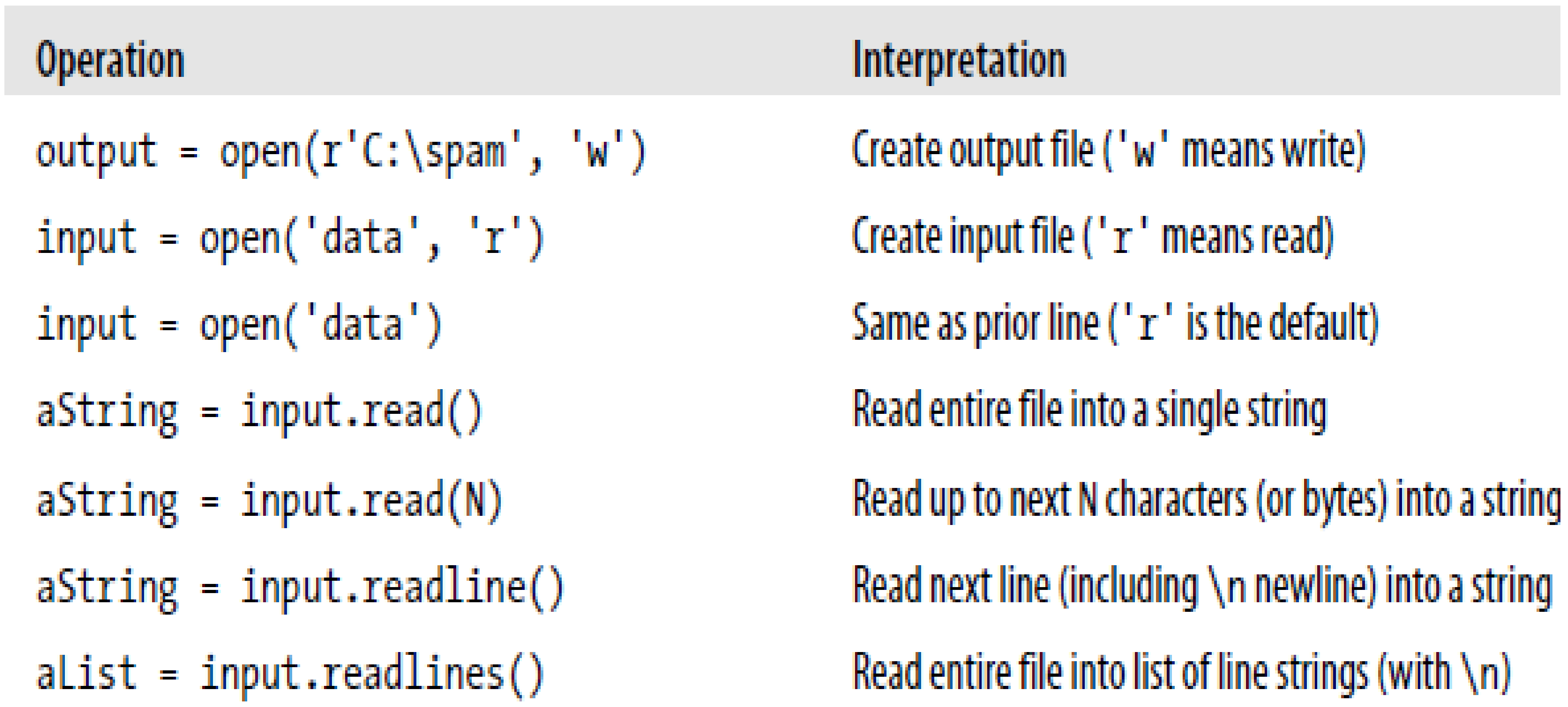
```python
f = open("1.txt")
f_content = f.readlines()
for line in f_content:
    print("---", line)
```

Output:



```
--- First line

--- Second line
```

File operations



Operation	Interpretation
`output = open(r'C:\spam', 'w')`	Create output file (`'w'` means write)
`input = open('data', 'r')`	Create input file (`'r'` means read)
`input = open('data')`	Same as prior line (`'r'` is the default)
`aString = input.read()`	Read entire file into a single string
`aString = input.read(N)`	Read up to next N characters (or bytes) into a string
`aString = input.readline()`	Read next line (including \n newline) into a string
`aList = input.readlines()`	Read entire file into list of line strings (with \n)

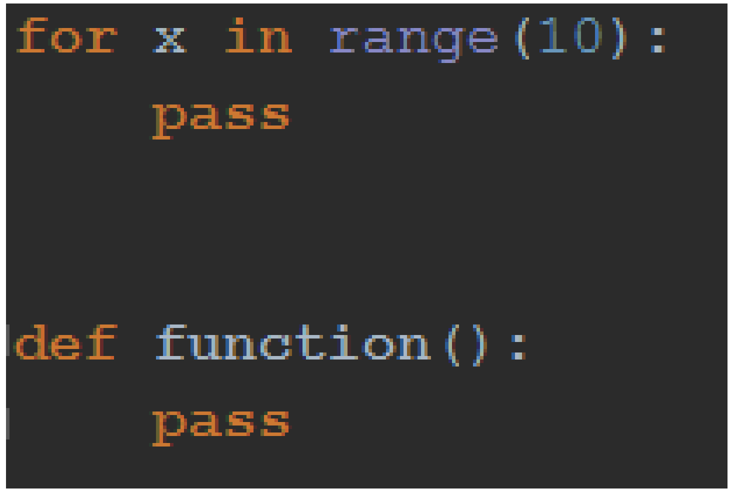
File operations

`output.write(aString)`	Write a string of characters (or bytes) into file
`output.writelines(aList)`	Write all line strings in a list into file
`output.close()`	Manual close (done for you when file is collected)
`output.flush()`	Flush output buffer to disk without closing
`anyFile.seek(N)`	Change file position to offset N for next operation
`for line in open('data'):` *`use line`*	File iterators read line by line
`open('f.txt', encoding='latin-1')`	Python 3.X Unicode text files (`str` strings)
`open('f.bin', 'rb')`	Python 3.X bytes files (`bytes` strings)
`codecs.open('f.txt', encoding='utf8')`	Python 2.X Unicode text files (`unicode` strings)
`open('f.bin', 'rb')`	Python 2.X bytes files (`str` strings)

Functions

```
def function(name):
    h = "Hello, " + name
    print(h)
    return h

function('Ann')
```

Empty blocks

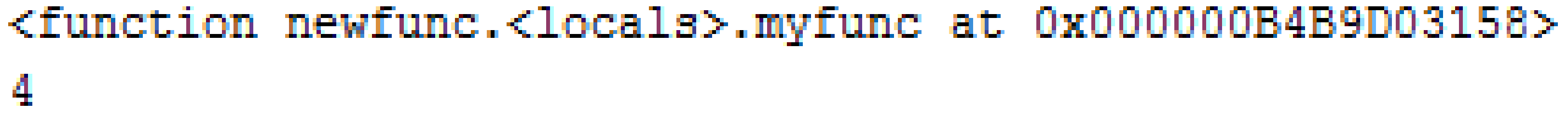


```
for x in range(10):
    pass


def function():
    pass
```

Functions

```
def newfunc(n):
    def myfunc(x):
        return x + n
    return myfunc

new = newfunc(2)
print(new)
print(new(2))
```

Output:



```
<function newfunc.<locals>.myfunc at 0x000000B4B9D03158>
4
```

Important functions

- max() / min()
  - sum()
  - abs()
  - random
- “Python”.title()
- “Python”.upper()
- “Python”.lower()