Python introduction

🐍

History

“I have this hope that there is a better way.

Higher-level tools that actually let you see the
structure of the software more clearly will be of tremendous value”

- Guido van Rossum,

  Python creator

Python

- High-level
- General-purpose
- Dynamic-typed
- Garbage-collected

Python

- **High-level**
- General-purpose
- Dynamic-typed
- Garbage-collected

Python

- High-level
- **General-purpose**
- Dynamic-typed
- Garbage-collected

Python

“If it walks like a duck, and it quacks like a duck, then it must be a duck”

- High-level
- General-purpose
- **Dynamic-typed**
- Garbage-collected

Python

- High-level
- General-purpose
- Dynamic-typed
- **Garbage-collected**

Python

- Interpreted ? ~~😇~~

Interpreted*

The *.py* source code is first compiled to **byte code** as *.pyc*

This byte code can be *interpreted* (official **CPython**), or JIT compiled (**PyPy**). Python source code (*.py*) can be compiled to different byte code also like **IronPython**(.Net) or **Jython** (JVM).

There are multiple implementations of **Python language** .

The official one is a **byte code** interpreted one.

* - most of the time

Hello world

```
>>> print("Hello world!")
Hello world!
```

Byte code

```
$ python3.7 -m dis example_hello.py

 1           0 LOAD_NAME                0 (print)
             2 LOAD_CONST               0 ('Hello world')
             4 CALL_FUNCTION            1
             6 POP_TOP
             8 LOAD_CONST               1 (None)
             10 RETURN_VALUE
```

Readability

```python
from os import listdir
from os.path import isfile

def get_files(path):
    result = []
    for entity in listdir(path):
        if isfile(entity):
            result.append(entity)
    return result

print(get_files('.'))
```

Readability

```python
from os import listdir
from os.path import isfile

def get_files(path):
    return [entity for entity in listdir(path) if isfile(entity)]

print(get_files('.'))
```

Builtin types

type

```
>>> type([1, 2, 3, 4])
<class 'list'>

>>> isinstance(10, int)
True
>>> isinstance(10, float)
False
```

None

- Represents an absence of a value
- Singleton
- Separate type **NoneType**
- Immutable

None

```
>>> type(None)
<class 'NoneType'>

>>> item_1 = None
>>> item_2 = None

>>> # Let's compare
>>> item_1 == item_2
True
```

None

```
>>> # For CPython - address
>>> id(None)
4551395432
>>> id(item_1)
4551395432
>>> id(item_2)
4551395432

>>> item_1 is item_2 # id(item_1) == id(item_2)
True
```

None

```
>>> none_item = print(None)
None

# All functions returns some value even if they don't
>>> print(none_item)
None
```

Numeric

- *integers*
- *floating point numbers*
- *complex numbers*

int

Integers have **unlimited** precision

```
>>> x = 2
>>> type(x)
<class 'int'>
>>> x ** 256
115792089237316195423570985008687907853269984665640564039457584
007913129639936
>>> x = 7
>>> x / 2 # quotient
3.5
>>> x // 2 # floored quotient
3
>>> x % 2 # remainder
1
>>> divmod(x, 2) # the pair (x // y, x % y)
(3, 1)
```

float

Floats are actually *double* in C

```
>>> x = 3.14
>>> type(x)
<class 'float'>


>>> x ** 2 # x to the power 2

9.8596

>>> x // 2  #floored quotient
1.0

>>> x % 2 # remainder
1.1400000000000001
```

float

```
>>> x = float("inf")
>>> x = float("-inf")

>>> x
inf

>>> type(x)
<class 'float'>
```

complex

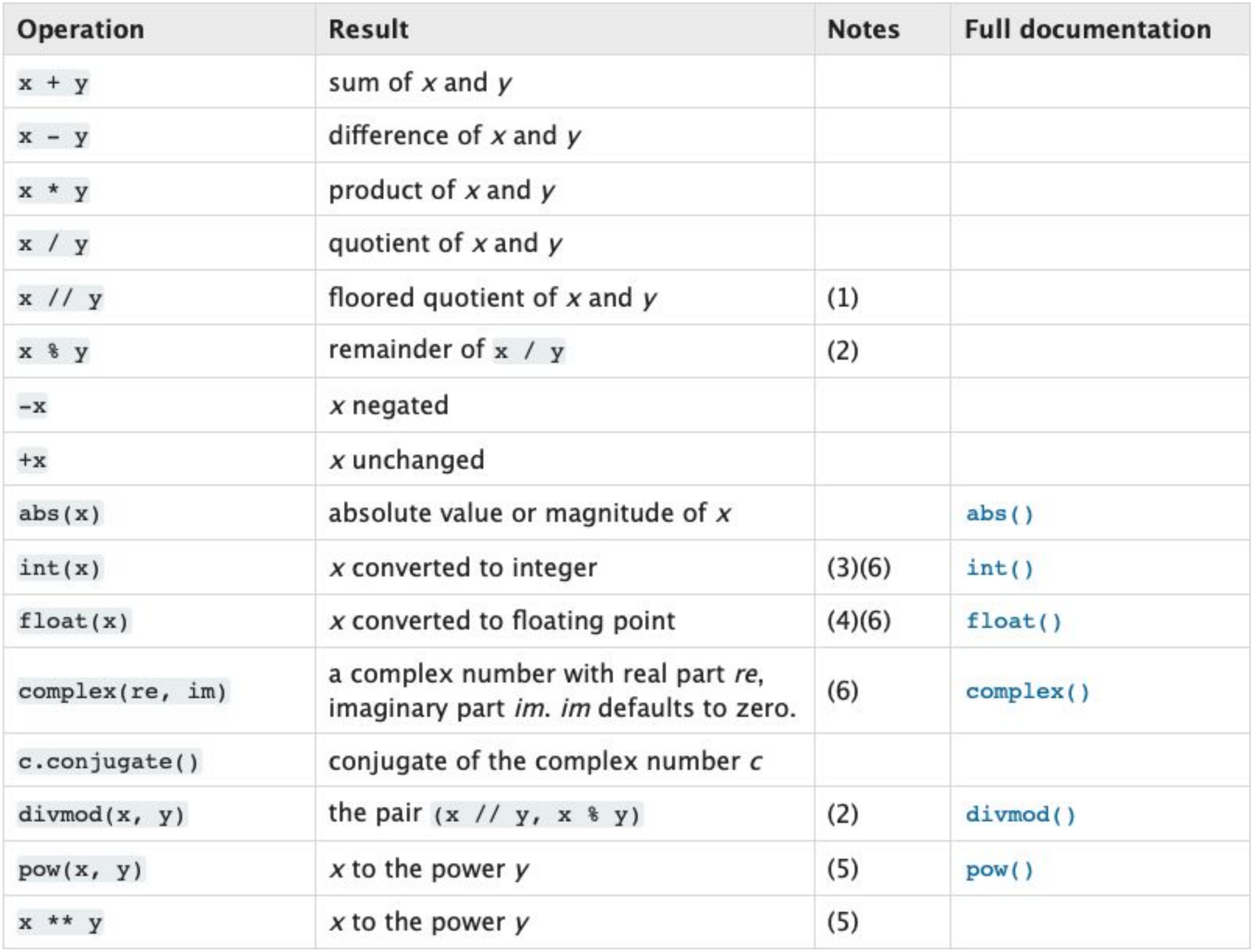
```
>>> z = 1.5 + 1.0j

>>> type(z)
<class 'complex'>

>>> z.real
1.5
>>> z.imag
1.0

>>> z1 = 1.5 - 1.0j
>>> z  /  z1
(0.3846153846153846+0.923076923076923j)
>>> x = complex('1.5+1j')

>>> x
(1.5+1j)
```



Operation	Result	Notes	Full documentation
`x + y`	sum of *x* and *y*		
`x - y`	difference of *x* and *y*		
`x * y`	product of *x* and *y*		
`x / y`	quotient of *x* and *y*		
`x // y`	floored quotient of *x* and *y*	(1)	
`x % y`	remainder of `x / y`	(2)	
`-x`	*x* negated		
`+x`	*x* unchanged		
`abs(x)`	absolute value or magnitude of *x*		`abs()`
`int(x)`	*x* converted to integer	(3)(6)	`int()`
`float(x)`	*x* converted to floating point	(4)(6)	`float()`
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.	(6)	`complex()`
`c.conjugate()`	conjugate of the complex number *c*		
`divmod(x, y)`	the pair `(x // y, x % y)`	(2)	`divmod()`
`pow(x, y)`	*x* to the power *y*	(5)	`pow()`
`x ** y`	*x* to the power *y*	(5)	

https://docs.python.org/3.7/library/stdtypes.html#numeric-types-int-float-complex

list

Mutable ordered sequence

Create list:

```
>>> lst = [1, 2, 3, 9, “str”, 7]
>>> lst = []
>>> lst = list()
>>> lst = [1, 2, 3, 9, 8, 7]

>>> new_lst = sorted(lst)
>>> new_lst
[1, 2, 3, 7, 8, 9]
>>> lst
[1, 2, 3, 9, 8, 7]

>>> lst.sort() # O(n * log n)
>>> lst
[1, 2, 3, 7, 8, 9]
```

list

Add new object in the end:

```
>>> lst = []
>>> lst.append(1) # O(1)
>>> lst
[1]
```

list

Concatenate:

```
>>> lst = [1]
>>> lst.extend([2, 3]) # O(m), m - size of concat list
>>> lst
[1, 2, 3]
```

Length:

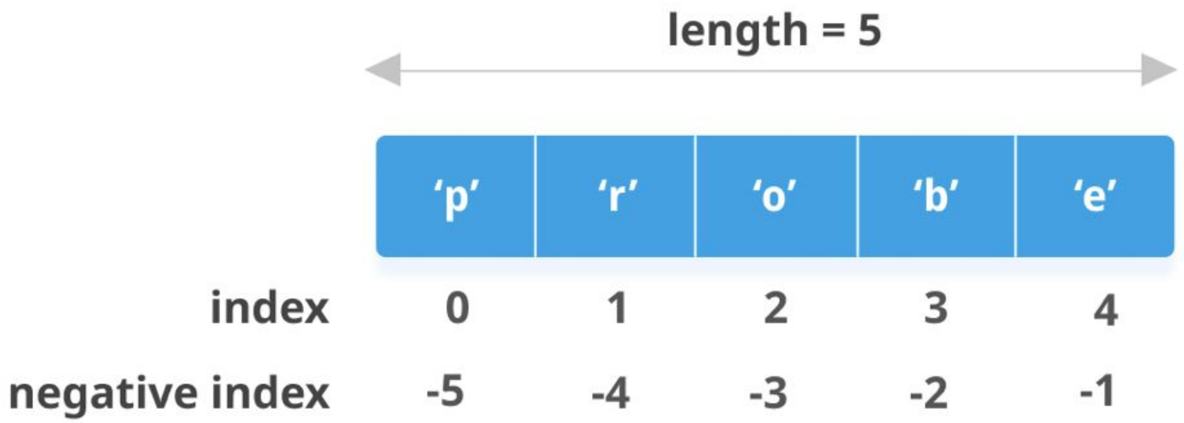
```
>>> len(lst) # O(1)
3
```

list

Element access:

```
>>> lst = [1, 2, 3]
>>> lst[0] # O(1)
1
>>> lst[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range

>>> lst[-1] # Last element
```



list

Insert:

```
>>> lst = [1, 2, 3]
>>> lst.insert(0, 42) # O(n)
>>> lst
[42, 1, 2, 3]
>>> lst.insert(300, 12)
>>> lst
[42, 1, 2, 3, 1, 12]
>>> lst[300]
```

list

Delete:

```
>>> del lst[0] # O(n)
>>> lst
[1, 2, 3, 1, 12]
```

Get last and delete:

```
>>> lst.pop() # For the last O(1)
12
```

Get lst[k] and delete:

```
>>> lst.pop(k) # For the last O(k)
12
```

list

Reverse:

```
>>> lst = [1, 2, 3, 4, 5, 6]
>>> lst.reverse() # O(n)
>>> lst
[6, 5, 4, 3, 2, 1]
```

Compare:

```
>>> l1 = [1, 2, 3]
>>> l2 = [1, 2, 3]
>>> l1 is l2
False
>>> l1 == l2 # O(n)
True
```

list

Containment:

```
>>> l1 = [1, 2, 3]
>>> 3 in l1 # O(n)
True
```

Multiplication:

```
>>> l1 = [1, 2, 3]
>>> l1 * 2
[1, 2, 3, 1, 2, 3]
```

Addition:

```
>>> l1 = [1, 2, 3]
>>> l2 = [3, 4, 5]
>>> l1 + l2
[1, 2, 3, 3, 4, 5]
```

str

Immutable sequence

```
>>> string = "one\ntwo\nthree\n"
>>> string.splitlines()
['one', 'two', 'three']

>>> type(string)
<class 'str'>

>>> string = "one, two, three"
>>> string.split()
['one,', 'two,', 'three']

>>> string.split(",")
['one', ' two', ' three']
```

str

```
>>> parts
['super', 'cali', 'fragilistic', 'expiali', 'docious']

>>> str()
''

>>> str().join(parts)
'supercalifragilisticexpialidocious'

>>> "".join(parts)
'supercalifragilisticexpialidocious'

>>> " ".join(parts)
'super cali fragilistic expiali docious'

>>> "+".join(parts)
'super+cali+fragilistic+expiali+docious'
```

str

```
>>> string = "Hello"

>>> string.islower()
False

>>> string.lower()
'hello'

>>> string.lower().islower()
True

>>> string.replace("He", "")
'llo'
```

str

```
>>> s = "Hello"

>>> string = f"{s} world"

>>> string
'Hello world'

>>> string = f"{12 + 1} sum"

>>> string
'13 sum’
```

bool

Singleton[s]

```
>>> a = True
>>> b = True
>>> a is b
True
>>> type(a)
<class 'bool'>

>>> isinstance(True, int)
True
>>> isinstance(True, bool)
True

>>> 1 + True
2
>>> int(True)
1
>>> int(False)
0
```

dict

- Dictionary in Python is ***Mapping***
- Dictionaries are indexed by ***keys***
- **Mutable**
- *From 3.6 ordered*

dict

Index:

```
>>> d = {'jack': 'white', 'black': 'jack'}
{'jack': 'white', 'black': 'jack'}

>>> d['jack']
'white'

>>> d['white']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'white'
```

dict

Get:

```
>>> d = {'jack': 'white', 'black': 'jack'}
{'jack': 'white', 'black': 'jack'}
>>> d['jack']
'white'
>>> d['white']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'white'

>>> d.get('white')
None

>>> d.get('white', 'no value')
'no value'
```

dict

Store:

```
>>> d = {'jack': 'white', 'black': 'jack'}
{'jack': 'white', 'black': 'jack'}
```

If key exists replace value
else creates key and store value

```
>>> d['white'] = 'jack' # O(1)
>>> d
{'jack': 'white', 'black': 'jack', 'white': 'jack'}
```

Length:

```
>>> len(d) # O(1)
3
```

dict

Delete:

```
>>> del d['white'] # O(1)
>>> d
{'jack': 'white', 'black': 'jack'}
```

Clear:

```
>>> d.clear() # O(1)
>>> d
{}
```

What to read?

- https://docs.python.org/3.7/
- https://docs.python-guide.org