

Python 3 – Quick Reference Card

Data types

Strings:

```
s = "foo bar"
s = 'foo bar'
s = r"c:\dir\new"          # raw (== 'c:\\dir\\new')
s = """Hello
    world"""
s.join(" baz")
n = len(s)
"Ala ma {} psy i {} koty".format(2,3)
"Square root of 2 is equal to {:.2f}".format(math.sqrt(2))
```

Lists:

```
L = [1, 2, 3, 4, 5]
L[0]          # single position
L[0:3]        # the first three elements
L[-2:]       # the last two elements
L[1:4] = [7,8] # substitute
del L[2]      # remove elements
L.append(x)   # x is a value
L.remove(x)
L.extend(L2)  # or: L3 = L + L2
L.pop()       # simple stack (with append)
L.sort()
x in L        # does L contain x?
L.index(x)    # index of the first occurrence
[x*2 for x in L if x>2] # list comprehensions
```

Tuples:

```
x = 1,2,3
x = (1,2,3)
x[1]
a,b,c = x
```

Dictionaries:

```
D = {'f1': 10, 'f2': 20}          # dict creation
D = dict(f1=10, f2=20)

keys = ('a', 'b', 'c')
D = dict.fromkeys(keys)          # new dict with empty values

for k in D: print(k)              # keys
for v in D.values(): print(v)    # values
for k, v in D.items():           # tuples with keys and values
list(D.keys())                   # list of keys
sorted(D.keys())                 # sorted list of keys

D = {}
```

```
D[(1,8,5)] = 100 # 3D sparse matrix
D.get((1,8,5))
D.get((1,1,1), -1)
```

Sets:

```
S = {1,3,5}
L = [1, 3, 1, 5, 3]
S = set(L) # set([1, 3, 5])
if 3 in S:
S1+S2, S1-S2, S1^S2, S1|S2
```

See also <https://docs.python.org/3/library/stdtypes.html>.

Loops

```
for x in range(6): # 0, 1, 2, 3, 4, 5
for x in range(1,6): # 1, 2, 3, 4, 5
for x in range(1,6,2): # 1, 3, 5

for k,v in D.items():
    print("D[{}]={}".format(k,v)) # D[f1]=10 D[f2]=20

L = [1, 3, 5]
for i,v in enumerate(L): # (index,value)
for x,y in zip(L1,L2): # returns tuples
for i in sorted(set(L)): print(i) # sorted set from a list
for x in reversed(L1):
```

Functions

```
def foo(arg1, *args, **dic):
    """Example documentation string.

    This function does not do anything special.
    """
    # arg1 is a positional argument
    # args is a list
    # dic is a dictionary of named arguments

def foo(a,b,c=0):
L = [1, 2, 3]
foo(*L) # unpacking a list of arguments
D = {'a': 10, 'b': 20}
foo(**D) # unpacking a dictionary of arguments

foo.__doc__ # the docstring
```

Input/output

Printing:

```
str(x) # human readable representation
repr(x) # interpretable representation
```

File access:

```
f = open("test.txt", "w")           # r / r+ / rb / rb+ / w / wb
f.write("Ala ma kota\n")
f.close()

for line in open("test.txt"): print(line, end="")

L = open("test.txt").readlines()    # returns a list of lines
```

Exclusive access:

```
f = os.fdopen(os.open("test.txt", os.O_WRONLY|os.O_EXCL), "w")
```

Input:

```
x = input("Name: ")
for line in sys.stdin: print(line)
```

String buffers:

```
from StringIO import StringIO
buf = StringIO()
sys.stdout = buf
print("Hello")
x = buf.getvalue()
```

Error stream:

```
print("Error!", file=sys.stderr, flush=True)
```

Other file operations:

os.rename(from, to)	os.remove(path)
os.chmod(file, 0700)	os.stat(file)

Special names

`__name__`

name of the file being run not imported

Typical usage:

```
if __name__ == "__main__":
    print("Do something")
```

Exceptions

```
try:
    raise TypeError("arg")
except (RuntimeError, NameError):
    pass                                # empty instruction (NOP)
except:
    info = sys.exc_info()
    print(info[0])
    print(info[1])
```

```

        traceback.print_tb(info[2])
    raise
else:
    ...
finally:
    ...
    # no exception but before finally
    # on the way out
    # unhandled exc, release resources

```

Object-oriented programming

```

class Person:
    ID = 0 # static variable
    def __init__(self, name, age=0):
        self.name = name
        self.age = age
        Person.ID += 1
        self.ID = Person.ID
    def lastName(self):
        return self.name.split()[-1]
    def __str__(self):
        return "{}({},{})".format(self.__class__.__name__,
                                   self.name, self.age)

class Worker(Person):
    def __init__(self, name, position, age=0):
        super().__init__(name, age)
        self.position = position
    def __str__(self):
        return "{}({},{},{})".format(self.__class__.__name__,
                                       self.name, self.position, self.age)

bob = Worker("Bob Smith", "developer", 25)
print(bob)

```

Useful APIs

Queues:

```

Q = collections.deque([10,20,30])
Q.append(40)
Q.popleft()

```

Pickling:

```

f = open("myobj.dat", "w")
pickle.dump(x, f)
f = open("myobj.dat", "r")
x = pickle.load(f)

```

Databases:

```

conn = sqlite3.connect("data.db")
c = conn.cursor()
c.execute("SELECT * FROM employees")
for row in c:
    print(row[0])
conn.commit()
conn.close()

```

```
db = shelve.open("file")
db["x"] = y
db.close()
```

CGI:

```
form = cgi.FieldStorage()
print("Content-type: text/html\n")
print(cgi.escape(form["user"].value))
```

HTTP Server:

```
srvraddr = ("", 8080) # my hostname, portnumber
srvrobject = BaseHTTPServer.HTTPServer(srvraddr,
                                         CGIHTTPServer.CGIHTTPRequestHandler)
srvrobject.serve_forever()
```

URLs:

```
conn = urllib.urlopen("http://localhost:8080")
reply = conn.read()
```

Environment

Encoding:

```
#!/usr/bin/python3
# -*- coding: latin-2 -*-
```

Windows – use **.pyw** extension to run the script (with GUI) without a console window.

Paths:

```
PYTHONPATH
export PYTHONSTARTUP=~/.pythonrc.py
```

Module **sys**:

```
sys.argv      sys.stdin      sys.stdout      sys.stderr
sys.path      sys.platform  sys.version
```

Processes (module **subprocess**):

```
res = subprocess.call(["hostname", "-f"], stderr=subprocess.DEVNULL)
res = subprocess.call("ps aux | grep ^root", shell=True)
output = subprocess.check_output(["mycmd", "myarg"], universal_newlines=True)
```

Module **os**:

```
os.pathsep    os.sep          os.pardir       os.curdir       os.linesep
os.startfile("index.html")
os.popen("ps ax").readlines()
os.listdir("/usr/local") # ['bin', 'etc', ...]
os.glob("*.txt")         # ['test.txt', 'out.txt', ...]
```

Module `os.path`:

```
os.path.split("/usr/bin/go.sh")      # ('/usr/bin', 'go.sh')
os.path.join("/usr/bin", "go.sh")   # '/usr/bin/go.sh'
os.path.splitext("/usr/bin/go.sh")  # ('/usr/bin/go', '.sh')
os.path.abspath("../bin/go.sh")     # '/usr/bin/go.sh'
os.path.isfile("go.sh")
```

Module `os.environ`:

```
os.environ.get("PYTHONSTARTUP")
```

Directories:

```
for (dir, subdirs, files) in os.walk("/tmp"):
    for f in files: print(f)
```

Functional programming

```
f = lambda x: x+10          # creates an anonymous function
f(5)                        # returns 15
L = [1, 4, 7]
for x in filter(lambda i: i<5, L): # returns [1, 4]
for x in map(lambda: x: x*2, L):   # returns [2, 8, 14]
```

References

1. The Python Tutorial, <https://docs.python.org/3/tutorial/>
2. The Python Language Reference, <http://docs.python.org/3/reference/>
3. The Python Standard Library, <http://docs.python.org/3/library/>
4. PEP-8: Style Guide for Python Code, <http://www.python.org/dev/peps/pep-0008/>