Cheat Sheet for Python: Life's Pathetic, Let's Pythonic!

Wenqiang Feng
E-mail: von198@gmail.com, Web: http://web.utk.edu/~wfeng1; Wrapped from python3-in-one-pic, © Copyrights belong to the original author.

The Zen of Python

```
import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
...
```

Syntax Roles

```
# Case-Seneitive
a = 1; A = 2
print(a is not A)  # True
# Comments
# Comments will be ignored # Comments will be ignored
# 4 spaces identation
# Code blocks are defined by
# their indentation
```

Native Datatypes

Number

Integer:

```
a = 1
b = 0x10 # 16
print(b) # 16
print(type(b)) # <class 'int'>
```

Float:

```
c = 1.2        # 1.2
d = .5         # 0.5
g = 3.14e-2     # 0.0314
print(type(g)) #<class 'float'>
```

Complex:

```
e = 1+2j         # (1+2j)
f = complex(1,2) # (1+2j)
print(type(e))#<class'complex'>
print(f == e)    # True
```

Operators:

```
print(1+1) # 2  print(2**2)# 4
print(2-2) # 0  print(9//4)# 2
print(3*3) # 9  print(9%4) # 1
print(5/4) # 1.25
```

Casting:

```
# Integer/String -> Float
print(float(3))      # 3.0
print(3/1)           # 3.0
print(float("3.14"))# 3.14
```

```
# Float/String -> Integer
int(3.14)                  # 3
int('100', base = 10)    # 100
int('1010', base = 2)    # 10
```

String

```
s1 = ':dog:\n Dogge'
s2 = "Dogge's home"
s3 = """
Hello,
Dogge!
"""
```

```
print(type(s1)) #<class 'str'>
print('%s, %s, %s'%(s1,s2,s3))
:dog:
 Dogge, Dogge's home,
Hello,
Dogge!
```

Length & operator:

```
print(len(s1))        # 12
print('ab'+'.'+'xy') # ab.xy
```

Slicing:

```
print('{0}:{1}' # Dogge:home
      .format(s2[:5],s2[-4:]))
```

Casting

```
print(str(3.14))     # 3.14
print(str(3))        # 3
print(str([1,2,3])) # [1,2,3]
print(str((1,2,3))) # (1,2,3)
```

```
print(str({1,2,3})) # {1,2,3}
print(str({'python':'py',
           'java':'js'}))
{'python':'*.py','java':'*.js'}
```

Native Datatypes

Boolean & None

```
True; False  # <class 'bool'>
```

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

List

```
l = ['python', 3, 'in', 'one']
```

```
print(type(l)) # <class 'list'>
```

Length:

```
print(len(l)) # 4
```

Index:

```
print(l.index(3)) # 1
```

Slicing:

```
print(l[0])
python
print(l[-1])
one
print(l[1:-1])
[3, 'in']
print(l[::-1])
['one', 'in', 3, 'python']
print(l[::2])
['python', 'in']
print(l[:-2:1])
['python', 3]
```

Alter:

```
l.pop() # 'one'
['python', 3, 'in']
print(l.pop(1)) # 'one'
3
print(l.remove('in')) # None
['python']
l.append('pic') # None
['python', 'pic']
l.extend(['!','!']) # None
['python', 'pic', '!', '!']
l.insert(2,4) # None
['python', 'pic', 4, '!', '!']
```

Tuple

```
tp=(1,2,3,[4,5])#Immutable list
```

```
print(type(l)) #<class 'tuple'>
```

Length & slicing & update:

```
print(len(tp)) # 4
print(tp[2])   # 3
print(tp[:3])  # (1,2,3)
tp[3][1] = 6   # (1,2,3,[4,6])
```

Assign multiple value:

```
v = (3,2,'a')
(c,b,a) = v
print(a,b,c)  # a 2 3
tp = (1,)      # Not tp=(1)
```

Set

```
st={'s','e','T'}
```

```
print(type(st)) #<class 'set'>
```

Length:

```
Unordered, collections, unique
print(len(st))       # 3
```

Empty:

```
st = set()
st.clear()
```

Alter

```
st.add('t') # {'e','T','t','s'}
st.add('t') # {'e','T','t','s'}
st.update(['!','!'])
# {'e', '!', 's', 'T', 't'}
```

```
st.pop() # {'!','s','T','t'}
st.discard('t') # {'!','s','T'}
st.remove('T')  # {'!', 's'}
st.clear() # set()
```

Dict

```
dic = {'k1': 1, 'k2': 2}
```

```
print(type(dic))#<class 'dict'>
```

Length & check:

```
print(len(dic))   # 2
print(dic.keys()) # ['k1','k2']
print(dic.values())# [1,2]
print('k2' in dic) # True
print('v1' in dic) # False
```

get & update:

```
print(dic['k1'])       # 1
print(dic.get('k1'))   # 1
dic['k2']= 3 #{'k1':1,'k2':3}
dic['k3']= 3
# {'k1':1, 'k2':3, 'k3': 3}
```

Flow Control

If

```
import sys
if sys.version_info.major < 3:
    print('Version 2.X')
elif sys.version_info.major > 3:
    print('Future')
else:
    print('Version 3.X')
```

Loop

For:

```
                        #  h
for i in 'hello':       #  e
    print(i)            #  l
                        #  l
                        #  o
```

While:

```
prod = 1; i =1
while i< 4:
    prod *= i
    i += 1
print(prod)       #  6
```

Break/continue:

```
for n in range(2, 10):
    if n % 2 ==0:
        print('even number',n)
        continue
    if n > 5:
        print('GT 5')
        break
```

Iterations & Generators:

```
python = iter('python')
<str_iterator object at ****>
for i in python:
    print(i)
def reverse(d):
    ix=range(len(d)-1,-1,-1)# n
    for i in ix:                # o
        yield d[i]              # h
nohtyp = reverse('python')  # t
for i in nohtyp:                # y
    print(i)                    # p
```

Comprehension

List

```
[2*x for x in range(4) if x**2>3]  # with filter [4, 6]
[4*x if x<2 else x for x in range(4)] # w/o filter [0, 4, 2, 3]
[(x,y) for x in range(2) for y in range(2)]
# [(0, 0), (0, 1), (1, 0), (1, 1)]
# matix transposed
matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
[[row[i] for row in matrix] for i in range(4)]
# [[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
[val for row in matrix for val in row ]
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Set

```
{2*x for x in range(4) if x**2>3}  # with filter {4, 6}
{4*x if x<2 else x for x in range(4)} # w/o filter {0, 2, 3, 4}
set([(x,y) for x in range(2) for y in range(2)])
# {(0, 1), (1, 0), (0, 0), (1, 1)}
```

Dict

```
ls = {s:len(s) for s in ['Python','Javascript','r']}
{'Python': 6, 'Javascript': 10, 'r': 1}
sl = {v: k for k, v in ls.items()}
{6: 'Python', 10: 'Javascript', 1: 'r'}
mapping = {'Python':'C','Javascript':'C++'}
{mapping.get(col,col):ls[col] for col in ls}
{'C': 6, 'C++': 10, 'r': 1}
```

Function

Definition

```
def func():
    """
    return 'Hello World!'
    """
    return 'Hello World!'
print(func())       Hello World!
print(func.__doc__) return 'Hello World!'
```

Default arguments

```
def func(name = 'George'):
    """
    return 'Hello World, name!'
    :param name: the name of the user. default: George
    :return: 'Hello World, name!'
    """
    return 'Hello World, {name}!'.format(name=name)
print(func())   Hello World, George!
```

Keyword arguments

```
def func(v, l = 'Python'):
    """
    return 'version, name!'
    """
    return '{v}, {l}!'.format(v=v, l=l)
print(func(3.6))  3.6, Python!
print(func(3.6,'r'))   3.6, r!
```

Arbitrary arguments

```
def func(*args, con=" & "):
    print(isinstance(args, tuple))
    print('Hello', con.join(args))
func('Python','C', 'C++')  Hello Python & C & C++
```

Lambda

```
pairs = [(1,'one'),(2,'two'),(3,'three'),(4,'four')]

pairs.sort(key=lambda pair: pair[1])
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
pairs.sort(key=lambda pair: pair[0],reverse=True)
[(4, 'four'), (3, 'three'), (2, 'two'), (1, 'one')]
```

Decorator

```
def log(f):
    def wrapper():
        print('Hey log~')
        f()
        print('Bye log!')
    return wrapper

@log
def fa():
    print('This is fa!')

# Equal to ...
def fb():
    print('This is fb!')

print(fa())  Hey log~ This is fa! Bye log! None
fb = log(fb) Hey log~This is fb! Bye log! None
```

Class (Object-oriented programming)

Class

```
class Animal:
    def fly(_):
        print('I can fly')

a = Animal()
a.fly()    # I can fly
print(a.__doc__) # This is an Animal.
```

__init__ & self

```
class Animal:
    def __init__(self, can_fly =False):
        self.can_fly  = can_fly
    def fly(self):
        if self.can_fly:
            print('I can fly')
        else:
            print('I can not fly')

a = Animal()            # callin g__init__() when instaniation!
a.fly()                 # I can not fly

b = Animal(can_fly=True) # callin g__init__() when instaniation!
b.fly()                 # I can fly
```

Instance

```
class Animal:
    pass
class Human:
    pass

a = Animal()
h = Human()
print(isinstance(a, Animal)) # True
print(isinstance(h,Animal))  # False
```

Inheritance

```
class Animal:
    def __init__(self, can_fly=False):
        self.can_fly  = can_fly
    def fly(self):
        if self.can_fly:
            print('I can fly')
        else:
            print('I can not fly')

class Dog(Animal):
    def bark(self):
        print('woof')
d = Dog()
d.fly()     # I can not fly
d.bark()    # woof
```

Override

```
class Bird(Animal):
    """
    This is a Dog.
    """
    def fly(self):
        print("I'm flying high!")
b = Bird()
b.fly()     # I'm flying high!
```

Module

Import

```
import os
from sys import version_info as PY_VERSION
print('version .'.format(PY_VERSION.major,PY_VERSION.minor))
# version 3.7
from math import pi
print(pi)     # 3.141592653589793
```

Path

```
%%script bash
pwd                   # /home/feng/Dropbox/MyPython/
echo $PYTHONPATH
/opt/spark/python:/opt/spark/python/lib/py4j-0.10.4-src.zip:
# python
import sys, os
os.path.abspath(' ')  # '/home/feng/Dropbox/MyPython/'
```

Package

```
MyModule
|-- SubModuleOne
    |-- __init__.py
    |-- smo.py
# smo.py
def run():
        print("Running MyModule.SubModuleOne.smo!")

from MyModule.SubModuleOne import smo
smo.run()   # Running MyModule.SubModuleOne.smo!
```

Pythonic

Reverse	`a[::-1]`
Check	`[s in J for s in S]   # J = 'aA', S = 'aAAbbb'`
Complex sum	`tuple(map(sum, zip(a,b))) # a = (1,0), b = (-1, 0)`
Transpose	`[[row[i] for row in M] for i in range(len(M[1]))]`
Flat list	`[val for row in matrix for val in row ]`
Exchange	`a, b = b, a`
With filter	`[2*x for x in range(4) if x**2>3]`
w/o filter	`[4*x if x<2 else x for x in range(4) ]`
dict get	`value = D.get(key, 0) # better than D[key]`

open file	`with open('filename.txt') as f:`
	`    for line in f:`
	`        print(line)`

string join	`''.join(letters)`

traverse indx	`for i, elem in enumerate(lst):`
	`        print(i,elem)`

zip	`for x, y in zip(a,b):`
	`    print(x,y)`

Dict traverse	`{v: k for k, v in D.items()}`
Counter	`Counter(s)   # from collections import Counter`
sorted	`sorted(d.items(), key=lambda t: t[1],reverse =True)`