Python cheatsheet

The Python cheat sheet is a one-page reference sheet for the Python 3 programming language.

Getting Started

Introduction

- Python (python.org)
- Learn X in Y minutes (learnxinyminutes.com)
- Regex in python (quickref.me)

Hello World

```
>>> print("Hello, World!")
Hello, World!
```

The famous "Hello World" program in Python

Variables

```
age = 18      # age is of type int
name = "John" # name is now of type str
print(name)
```

Python can't declare a variable without assignment.

Data Types

`str`	Text
`int`, `float`, `complex`	Numeric
`list`, `tuple`, `range`	Sequence
`dict`	Mapping
`set`, `frozenset`	Set
`bool`	Boolean
`bytes`, `bytearray`, `memoryview`	Binary

See: Data Types

Slicing String

```
>>> msg = "Hello, World!"
>>> print(msg[2:5])
llo
```

See: Strings

Lists

```
mylist = []
mylist.append(1)
mylist.append(2)
for item in mylist:
    print(item) # prints out 1,2
```

See: Lists

If Else

```
num = 200
if num > 0:
    print("num is greater than 0")
else:
    print("num is not greater than 0")
```

See: Flow control

Loops

```
for item in range(6):
    if item == 3: break
    print(item)
else:
    print("Finally finished!")
```

See: Loops

Functions

```
>>> def my_function():
...     print("Hello from a function")
...
>>> my_function()
Hello from a function
```

See: Functions

File Handling

```
with open("myfile.txt", "r", encoding='utf8') as file:
    for line in file:
        print(line)
```

See: File Handling

Arithmetic

```
result = 10 + 30 # => 40
result = 40 - 10 # => 30
result = 50 * 5  # => 250
result = 16 / 4  # => 4.0 (Float Division)
result = 16 // 4 # => 4 (Integer Division)
result = 25 % 2  # => 1
result = 5 ** 3  # => 125
```

The `/` means quotient of x and y, and the `//` means floored quotient of x and y, also see StackOverflow

Plus-Equals

```
counter = 0
counter += 10           # => 10
counter = 0
counter = counter + 10  # => 10

message = "Part 1."

# => Part 1.Part 2.
message += "Part 2."
```

f-Strings (Python 3.6+)

```
>>> website = 'Quickref.ME'
>>> f"Hello, {website}"
"Hello, Quickref.ME"

>>> num = 10
>>> f'{num} + 10 = {num + 10}'
'10 + 10 = 20'
```

See: Python F-Strings

Python Built-in Data Types

Strings

```
hello = "Hello World"
hello = 'Hello World'

multi_string = """Multiline Strings
Lorem ipsum dolor sit amet,
consectetur adipiscing elit """
```

See: Strings

Numbers

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

>>> print(type(x))
<class 'int'>
```

Booleans

```
my_bool = True
my_bool = False

bool(0)     # => False
bool(1)     # => True
```

Lists

Tuple

Set

```
list1 = ["apple", "banana", "cherry"]
list2 = [True, False, False]
list3 = [1, 5, 7, 9, 3]
list4 = list((1, 5, 7, 9, 3))
```

See: Lists

```
my_tuple = (1, 2, 3)
my_tuple = tuple((1, 2, 3))
```

Similar to List but immutable

```
set1 = {"a", "b", "c"}
set2 = set(("a", "b", "c"))
```

Set of unique items/objects

Dictionary

```
>>> empty_dict = {}
>>> a = {"one": 1, "two": 2, "three": 3}
>>> a["one"]
1
>>> a.keys()
dict_keys(['one', 'two', 'three'])
>>> a.values()
dict_values([1, 2, 3])
>>> a.update({"four": 4})
>>> a.keys()
dict_keys(['one', 'two', 'three',
'four'])
>>> a['four']
4
```

Key: Value pair, JSON like object

Casting

Integers

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Floats

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Strings

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Python Advanced Data Types

Heaps

```
import heapq

myList = [9, 5, 4, 1, 3, 2]
heapq.heapify(myList) # turn myList into a Min Heap
print(myList)    # => [1, 3, 2, 5, 9, 4]
print(myList[0]) # first value is always the smallest in the heap

heapq.heappush(myList, 10) # insert 10
x = heapq.heappop(myList)  # pop and return smallest item
print(x)                   # => 1
```

Negate all values to use Min Heap as Max Heap

```
myList = [9, 5, 4, 1, 3, 2]
myList = [-val for val in myList] # multiply by -1 to negate
heapq.heapify(myList)

x = heapq.heappop(myList)
print(-x) # => 9 (making sure to multiply by -1 again)
```

Heaps are binary trees for which every parent node has a value less than or equal to any of its children. Useful for accessing min/max value quickly. Time complexity: O(n) for heapify, O(log n) push and pop. See: Heapq

Stacks and Queues

```
from collections import deque

q = deque()          # empty
q = deque([1, 2, 3]) # with values

q.append(4)     # append to right side
q.appendleft(0) # append to left side
print(q)    # => deque([0, 1, 2, 3, 4])

x = q.pop() # remove & return from right
y = q.popleft() # remove & return from lef
print(x)    # => 4
print(y)    # => 0
print(q)    # => deque([1, 2, 3])

q.rotate(1) # rotate 1 step to the right
print(q)    # => deque([3, 1, 2])
```

Deque is a double-ended queue with O(1) time for append/pop operations from both sides. Used as stacks and queues. See: Deque

Python Strings

Array-like

```
>>> hello = "Hello, World"
>>> print(hello[1])
e
>>> print(hello[-1])
d
```

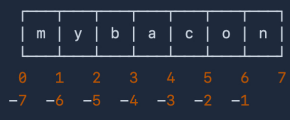
Get the character at position 1 or last

Looping

```
>>> for char in "foo":
...     print(char)
f
o
o
```

Loop through the letters in the word "foo"

Slicing string




```
>>> s = 'mybacon'
>>> s[2:5]
'bac'
>>> s[0:2]
'my'

>>> s = 'mybacon'
>>> s[:2]
'my'
>>> s[2:]
'bacon'
>>> s[:2] + s[2:]
'mybacon'
>>> s[:]
'mybacon'

>>> s = 'mybacon'
>>> s[-5:-1]
```

String Length

```
>>> hello = "Hello, World!"
>>> print(len(hello))
13
```

The len() function returns the length of a string

Multiple copies

```
>>> s = '===+'
>>> n = 8
>>> s * n
'===+===+===+===+===+===+===+===+'
```

Check String

```
>>> s = 'spam'
>>> s in 'I saw spamalot!'
True
>>> s not in 'I saw The Holy Grail!'
True
```

Concatenates

```
>>> s = 'spam'
>>> t = 'egg'
>>> s + t
'spamegg'
>>> 'spam' 'egg'
'spamegg'
```

```
'baco'
>>> s[2:6]
'baco'
```

Formatting

```
name = "John"
print("Hello, %s!" % name)

name = "John"
age = 23
print("%s is %d years old." % (name, age))
```

format() Method

```
txt1 = "My name is {fname}, I'm {age}".format(fname="John", age=36)
txt2 = "My name is {0}, I'm {1}".format("John", 36)
txt3 = "My name is {}, I'm {}".format("John", 36)
```

With a stride

```
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::5]
'11111'
>>> s[4::5]
'55555'
>>> s[::-5]
'55555'
>>> s[::-1]
'5432154321543215432154321'
```

Input

```
>>> name = input("Enter your name: ")
Enter your name: Tom
>>> name
'Tom'
```

Get input data from console

Join

```
>>> "#".join(["John", "Peter", "Vicky"])
'John#Peter#Vicky'
```

Endswith

```
>>> "Hello, world!".endswith("!")
True
```

Python F-Strings (Since Python 3.6+)

f-Strings usage

```
>>> website = 'Quickref.ME'
>>> f"Hello, {website}"
"Hello, Quickref.ME"

>>> num = 10
>>> f'{num} + 10 = {num + 10}'
'10 + 10 = 20'

>>> f"""He said {"I'm John"}"""
"He said I'm John"

>>> f'5 {"{stars}"}'
'5 {stars}'
>>> f'{{5}} {"stars"}'
'{5} stars'

>>> name = 'Eric'
>>> age = 27
>>> f"""Hello!
...     I'm {name}.
...     I'm {age}."""
"Hello!\n    I'm Eric.\n    I'm 27."
```

it is available since Python 3.6, also see: Formatted string literals

f-Strings Fill Align

```
>>> f'{"text":10}'     # [width]
'text      '
>>> f'{"test":*>10}'   # fill left
'******test'
>>> f'{"test":*<10}'   # fill right
'test******'
>>> f'{"test":*^10}'   # fill center
'***test***'
>>> f'{12345:0=10}'    # fill with numbers
'0000012345'
```

f-Strings Type

```
>>> f'{10:b}'        # binary type
'1010'
>>> f'{10:o}'        # octal type
'12'
>>> f'{200:x}'       # hexadecimal type
'c8'
>>> f'{200:X}'
'C8'
>>> f'{345600000000:e}' # scientific notat
'3.456000e+11'
>>> f'{65:c}'        # character type
'A'
>>> f'{10:#b}'       # [type] with notatior
'0b1010'
>>> f'{10:#o}'
'0o12'
>>> f'{10:#x}'
'0xa'
```

F-Strings Others

```
>>> f'{-12345:0=10}'  # negative numbers
'-000012345'
>>> f'{12345:0}'      # [0] shortcut (no a
'0000012345'
>>> f'{-12345:0}'
'-000012345'
>>> import math       # [.precision]
>>> math.pi
3.141592653589793
>>> f'{math.pi:.2f}'
'3.14'
>>> f'{1000000:,.2f}' # [grouping_option]
'1,000,000.00'
>>> f'{1000000:_.2f}'
'1_000_000.00'
>>> f'{0.25:0%}'      # percentage
'25.000000%'
>>> f'{0.25:.0%}'
'25%'
```

F-Strings Sign

```
>>> f'{12345:+}'      # [sign] (+/-)
'+12345'
>>> f'{-12345:+}'
'-12345'
>>> f'{-12345:10}'
'    -12345'
>>> f'{-12345:=10}'
'-000012345'
```

Python Lists

Defining

Generate

```
>>> li1 = []
>>> li1
[]
>>> li2 = [4, 5, 6]
>>> li2
[4, 5, 6]
>>> li3 = list((1, 2, 3))
>>> li3
[1, 2, 3]
>>> li4 = list(range(1, 11))
>>> li4
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> list(filter(lambda x : x % 2 == 1, range(1, 20)))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

>>> [x ** 2 for x in range (1, 11) if  x % 2 == 1]
[1, 9, 25, 49, 81]

>>> [x for x in [3, 4, 5, 6, 7] if x > 5]
[6, 7]

>>> list(filter(lambda x: x > 5, [3, 4, 5, 6, 7]))
[6, 7]
```

Append

```
>>> li = []
>>> li.append(1)
>>> li
[1]
>>> li.append(2)
>>> li
[1, 2]
>>> li.append(4)
>>> li
[1, 2, 4]
>>> li.append(3)
>>> li
[1, 2, 4, 3]
```

Remove

```
>>> li = ['bread', 'butter', 'milk']
>>> li.pop()
'milk'
>>> li
['bread', 'butter']
>>> del li[0]
>>> li
['butter']
```

Access

```
>>> li = ['a', 'b', 'c', 'd']
>>> li[0]
'a'
>>> li[-1]
'd'
>>> li[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

List Slicing

Syntax of list slicing:

```
a_list[start:end]
a_list[start:end:step]
```

Slicing

```
>>> a = ['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[2:5]
['bacon', 'tomato', 'ham']
>>> a[-5:-2]
['egg', 'bacon', 'tomato']
>>> a[1:4]
['egg', 'bacon', 'tomato']
```

Omitting index

```
>>> a[:4]
['spam', 'egg', 'bacon', 'tomato']
>>> a[0:4]
['spam', 'egg', 'bacon', 'tomato']
>>> a[2:]
['bacon', 'tomato', 'ham', 'lobster']
>>> a[2:len(a)]
['bacon', 'tomato', 'ham', 'lobster']
>>> a
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[:]
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
```

With a stride

```
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[0:6:2]
['spam', 'bacon', 'ham']
>>> a[1:6:2]
['egg', 'tomato', 'lobster']
>>> a[6:0:-2]
['lobster', 'tomato', 'egg']
>>> a
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[::-1]
['lobster', 'ham', 'tomato', 'bacon', 'egg', 'spam']
```

Concatenating

```
>>> odd = [1, 3, 5]
>>> odd.extend([9, 11, 13])
>>> odd
[1, 3, 5, 9, 11, 13]
>>> odd = [1, 3, 5]
>>> odd + [9, 11, 13]
[1, 3, 5, 9, 11, 13]
```

Sort & Reverse

```
>>> li = [3, 1, 3, 2, 5]
>>> li.sort()
>>> li
[1, 2, 3, 3, 5]
>>> li.reverse()
>>> li
[5, 3, 3, 2, 1]
```

Count

```
>>> li = [3, 1, 3, 2, 5]
>>> li.count(3)
2
```

Repeating

```
>>> li = ["re"] * 3
>>> li
['re', 're', 're']
```

Python Flow control

Basic

```
num = 5
if num > 10:
    print("num is totally bigger than 10."
elif num < 10:
    print("num is smaller than 10.")
else:
    print("num is indeed 10.")
```

One line

```
>>> a = 330
>>> b = 200
>>> r = "a" if a > b else "b"
>>> print(r)
a
```

else if

```
value = True
if not value:
    print("Value is False")
elif value is None:
    print("Value is None")
else:
    print("Value is True")
```

Python Loops

Basic

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

Prints: 2 3 5 7

With index

```
animals = ["dog", "cat", "mouse"]
# enumerate() adds counter to an iterable
for i, value in enumerate(animals):
    print(i, value)
```

Prints: 0 dog 1 cat 2 mouse

While

```
x = 0
while x < 4:
    print(x)
    x += 1  # Shorthand for x = x + 1
```

Prints: 0 1 2 3

Break

```
x = 0
for index in range(10):
    x = index * 10
    if index == 5:
        break
    print(x)
```

Prints: 0 10 20 30 40

Continue

```
for index in range(3, 8):
    x = index * 10
    if index == 5:
        continue
    print(x)
```

Prints: 30 40 60 70

Range

```
for i in range(4):
    print(i) # Prints: 0 1 2 3

for i in range(4, 8):
    print(i) # Prints: 4 5 6 7

for i in range(4, 10, 2):
    print(i) # Prints: 4 6 8
```

With zip()

```
words = ['Mon', 'Tue', 'Wed']
nums = [1, 2, 3]
# Use zip to pack into a tuple list
for w, n in zip(words, nums):
    print('%d:%s, ' %(n, w))
```

Prints: 1:Mon, 2:Tue, 3:Wed,

for/else

```
nums = [60, 70, 30, 110, 90]
for n in nums:
    if n > 100:
        print("%d is bigger than 100" %n)
        break
else:
    print("Not found!")
```

Also see: Python Tips

Python Functions

Basic

```
def hello_world():
    print('Hello, World!')
```

Return

```
def add(x, y):
    print("x is %s, y is %s" %(x, y))
    return x + y

add(5, 6)    # => 11
```

Positional arguments

```
def varargs(*args):
    return args

varargs(1, 2, 3)  # => (1, 2, 3)
```

Keyword arguments

```
def keyword_args(**kwargs):
    return kwargs

# => {"big": "foot", "loch": "ness"}
keyword_args(big="foot", loch="ness")
```

Returning multiple

```
def swap(x, y):
    return y, x

x = 1
y = 2
x, y = swap(x, y)  # => x = 2, y = 1
```

Default Value

```
def add(x, y=10):
    return x + y

add(5)      # => 15
add(5, 20)  # => 25
```

Anonymous functions

```
# => True
(lambda x: x > 2)(3)

# => 5
(lambda x, y: x ** 2 + y ** 2)(2, 1)
```

Python Modules

Import modules

```
import math
print(math.sqrt(16))  # => 4.0
```

From a module

```
from math import ceil, floor
print(ceil(3.7))   # => 4.0
print(floor(3.7))  # => 3.0
```

Import all

```
from math import *
```

Shorten module

```
import math as m

# => True
math.sqrt(16) == m.sqrt(16)
```

Functions and attributes

```
import math
dir(math)
```

Python File Handling

Read file

Line by line

```
with open("myfile.txt") as file:
    for line in file:
        print(line)
```

With line number

```
file = open('myfile.txt', 'r')
for i, line in enumerate(file, start=1):
    print("Number %s: %s" % (i, line))
```

String

Write a string

```
contents = {"aa": 12, "bb": 21}
with open("myfile1.txt", "w+") as file:
    file.write(str(contents))
```

Read a string

```
with open('myfile1.txt', "r+") as file:
    contents = file.read()
print(contents)
```

Object

Write an object

```
contents = {"aa": 12, "bb": 21}
with open("myfile2.txt", "w+") as file:
    file.write(json.dumps(contents))
```

Read an object

```
with open('myfile2.txt', "r+") as file:
    contents = json.load(file)
print(contents)
```

Delete a File

```
import os
os.remove("myfile.txt")
```

Check and Delete

```
import os
if os.path.exists("myfile.txt"):
    os.remove("myfile.txt")
else:
    print("The file does not exist")
```

Delete Folder

```
import os
os.rmdir("myfolder")
```

Python Classes & Inheritance

Defining

```
class MyNewClass:
    pass

# Class Instantiation
my = MyNewClass()
```

Constructors

```
class Animal:
    def __init__(self, voice):
        self.voice = voice

cat = Animal('Meow')
print(cat.voice)    # => Meow

dog = Animal('Woof')
print(dog.voice)    # => Woof
```

Method

```
class Dog:

    # Method of the class
    def bark(self):
        print("Ham-Ham")

charlie = Dog()
charlie.bark()   # => "Ham-Ham"
```

Class Variables

```
class MyClass:
    class_variable = "A class variable!"

# => A class variable!
print(MyClass.class_variable)

x = MyClass()

# => A class variable!
print(x.class_variable)
```

Super() Function

```
class ParentClass:
    def print_test(self):
        print("Parent Method")

class ChildClass(ParentClass):
    def print_test(self):
        print("Child Method")
        # Calls the parent's print_test()
        super().print_test()
```

```
>>> child_instance = ChildClass()
>>> child_instance.print_test()
Child Method
Parent Method
```

repr() method

```
class Employee:
    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return self.name

john = Employee('John')
print(john)  # => John
```

User-defined exceptions

```
class CustomError(Exception):
    pass
```

Polymorphism

```
class ParentClass:
    def print_self(self):
        print('A')

class ChildClass(ParentClass):
    def print_self(self):
        print('B')

obj_A = ParentClass()
obj_B = ChildClass()

obj_A.print_self() # => A
obj_B.print_self() # => B
```

Overriding

```
class ParentClass:
    def print_self(self):
        print("Parent")

class ChildClass(ParentClass):
    def print_self(self):
        print("Child")

child_instance = ChildClass()
child_instance.print_self() # => Child
```

Inheritance

```
class Animal:
    def __init__(self, name, legs):
        self.name = name
        self.legs = legs

class Dog(Animal):
    def sound(self):
        print("Woof!")

Yoki = Dog("Yoki", 4)
print(Yoki.name) # => YOKI
print(Yoki.legs) # => 4
Yoki.sound()     # => Woof!
```

Miscellaneous

Comments

```
# This is a single line comments.

""" Multiline strings can be written
    using three "s, and are often used
    as documentation.
"""
```

Generators

```
def double_numbers(iterable):
    for i in iterable:
        yield i + i
```

Generator to list

```
values = (-x for x in [1,2,3,4,5])
gen_to_list = list(values)

# => [-1, -2, -3, -4, -5]
print(gen_to_list)
```

```
''' Multiline strings can be written
    using three 's, and are often used
    as documentation.
'''
```

Generators help you make lazy code.

Handle exceptions

```
try:
    # Use "raise" to raise an error
    raise IndexError("This is an index error")
except IndexError as e:
    pass                # Pass is just a no-op. Usually you would do recovery here.
except (TypeError, NameError):
    pass                # Multiple exceptions can be handled together, if required.
else:                   # Optional clause to the try/except block. Must follow all except blocks
    print("All good!")   # Runs only if the code in try raises no exceptions
finally:                # Execute under all circumstances
    print("We can clean up resources here")
```

Related Cheatsheet

- **Awk** Cheatsheet Quick Reference
- **Bash** Cheatsheet Quick Reference

Recent Cheatsheet

- **Google Search** Cheatshee Quick Reference
- **Kubernetes** Cheatsheet Quick Reference
- **ES6** Cheatsheet Quick Reference
- **ASCII Code** Cheatsheet Quick Reference

QuickRef.ME

Share quick reference and cheat sheet for developers.

中文版 #Notes