Python Cheat Sheet

(Cover all Basic Python Syntaxes)

Over 300 Examples

Ray Yao (2nd Edition)

Python

Cheat Sheet

**(Cover all Basic Python Syntaxes,
More Than 300 Examples)**

Ray Yao

About This Book

This book covers all basic Python syntaxes**.** We can quickly reference the most helpful programming syntaxes, such as common command syntax, string function syntax, collection function syntax, class & object syntax……; all these syntaxes are very useful for programming**.**
We can take this book as a basic syntax manual because its entries are arranged alphabetically so that we can easily reference the important syntax**.**
Nowadays or in the future, the Python Syntax Book can provide great help for coding both in our study and our work**.**

Disclaimer

This book is intended as a basic syntax manual only; it cannot include all entries on this subject**.** Its purpose is as a supplement for a cheat sheet book, not as a whole Python dictionary**.**

Table of Contents

Syntax Chart

 # symbol syntax:

#

This symbol is used as a comment symbol.

e.g.

print("Hello World! ") # print() is used to output contents

Output: Hello World!

*** symbol syntax:**

*

Match 0 or more characters in regular expression.

e.g.

re*

Match 0 or more characters, such as red, read, reader……

**** symbol syntax:**

**

Exponentiation

e.g.

a = 2

b = 3

print(a ** b) # just like 2*2*2

Output: 8

+ symbol syntax:

+

Concatenate two strings together

e.g.

myString = "Python "+ "is a good language"

Output: Python is a good language

// symbol syntax:

//

Floor division: rounds the result down to the closest number

e.g.

```
a = 17
b = 2
print(a // b)
```

Output: 8

>>> symbol syntax:

>>>

This symbol is the Python interactive command shell prompt, requests the input from the user.

e.g.

```
>>> " Hello " + " World! "
```

Output: Hello World!

''' symbol syntax:

'''

multi-line strings

'''

Three single quotes are used for multi-line string

e.g.

```
str = ''' We are a multi-line string,
which contains more than one line strings.
The multi-line string can also work as a comment sometimes.
'''
print(str)
```

Output:

We are a multi-line string,
which contains more than one line strings.
The multi-line string can also work as a comment sometimes.

""" symbol syntax:

"""

multi-line strings

"""
Three double quotes are used for multi-line string
e.g.
str = """ We are a multi-line string,
which contains more than one line strings.
The multi-line string can also work as a comment sometimes.
"""
print(str)
# Output:
We are a multi-line string,
which contains more than one line strings.
The multi-line string can also work as a comment sometimes.

__del__() syntax:

```
def __del__(self):
```

Define a destructor method, which is called when all object resources are deleted.
e.g.

```
class MyClass:
    def __init__(self):
        print("Constructor is called")
    def __del__(self):
        print("Destructor is called")
obj = MyClass()
del obj
```

Output:
Constructor is called
Destructor is called

__format__() syntax:

```
def __format__(self, myFormat):
```

Define a function to format the specified string
e.g.

```
class MyClass:
    def __format__(self, myFormat):
        if myFormat == 'No good!':
```

```
            return 'Very good!'
        return myFormat
c = MyClass()
print(format(c, 'No good!'))
# Output:  Very good!
```

__import__() syntax:

```
__import__('module')
# Import a module to the current file.
e.g.
datetime = __import__('datetime')    # import datetime module
dt=datetime.datetime.now()
print(dt.__str__())
# Output:  2023-03-30 12:57:26.678333
```

__init__() syntax:

```
def __init__(self, var1, var2,…):
# Define a function to initialize the class's variable when a class is created
e.g.
class Student:
    def __init__(self, name, id):
        self.name = name
        self.id = id
s = Student("Smith", "St007")
print(s.name)
print(s.id)
# Output:    Smith    St007
```

__len__() syntax:

```
object.__len__()
# Define a function to get an integer that represents the object length.
e.g.
class MyClass:
    def __len__(self):
        return 100
cls = MyClass()
```

```
print(len(cls))
# Output:  100
```

__main__ syntax:

```
__name__ == "__main__":
# Return True  if the program is run directly by interpreter.
# Return False if the program is imported as a module
e.g.
if __name__ == "__main__":
    print ("Executed if the program is run directly by interpreter")
else:
    print ("Executed if the program is imported as a module")
# Output:
Executed if the program is run directly by interpreter
```

__missing_ () syntax:

```
def __missing__(self, key):
# Define a function, provide the default value to a dictionary key.
e.g.
class MyClass(dict):
    def __missing__(self, key):
        return 'Sorry, the age is not found!'
myDict = MyClass({'Andy': 16, 'Betty': 17, 'Cindy': 18})
print(myDict['Davy'])
# Output:  Sorry, the age is not found!
```

__new__() syntax:

```
class ClassName:
    def __new__(cls):
        return super(ClassName, cls).__new__(cls)
# __new__() is the constructor method. It is called first and it returns a new class instance 'cls'. Namely it is used to create a new instance of a class 'cls' and takes as the first argument in the class.
# __new__() is always called before __init__()
e.g.
class MyClass(object):
```

```
    def __new__(cls):
        print("Create new class instance")
        return super(MyClass, cls).__new__(cls)
    def __init__(self):
        print("__init__() is called")
MyClass()
# Output:
Create a new class instance
__init__() is called
```

__repr__() syntax:

object.__repr__()

Call __repr__() when an object is printed, __repr__() returns a string representation of an object, the string may contain more information than __str__().

e.g.

```
import datetime
dt=datetime.datetime.now()
print(dt.__repr__())
# Output:  datetime.datetime(2023, 3, 30, 13, 20, 15, 404708)
```

__str__() syntax:

object.__str__()

Call __str__() when an object is printed, __str__() returns a string representation of an object.

e.g.

```
import datetime
dt=datetime.datetime.now()
print(dt.__str__())
# Output:  2023-03-30 12:57:26.678333
```

---

abs() syntax:

abs( )

Return an absolute value of a number.

e.g.

```
print("abs(-100) =", abs(-100))
```

```
# Output:  abs(-100) = 100
```

---

access() syntax:

os.access(path, mode)

```
# Access a path/file, return a "mode" to check the status of path/file.
# "mode" has four parameters as follows:
1. os.F_OK: check if the path/file exists.
2. os.R_OK: check if the path/file can be read.
3. os.W_OK: check if the path/file can be written.
4. os.X_OK: check if path/file can be executed.
e.g.
import os
import sys
file1 = os.access("myfile.txt", os.F_OK)
print("The file exists?:", file1)
file2 = os.access("myfile.txt", os.R_OK)
print("The file can be read?:", file2)
file3 = os.access("myfile.txt", os.W_OK)
print("The file can be written?:", file3)
file4 = os.access("myfile.txt", os.X_OK)
print("The file can be executed?:", file4)
# Output:
The file exists?: True
The file can be read?: True
The file can be written?: True
The file can be executed?: True
```

---

add() syntax:

set.add("element")

```
# Add an element to the unsorted set
e.g.
mySet = {"ant", "bee", "cat"}
mySet.add("dog")
print(mySet)
# Output:  {'cat', 'dog', 'ant', 'bee'}
```

---

all() syntax:

all()

Return true if all elements in a collection are true

e.g.

```
mylist = [True, False, True]
print(all(mylist))
```

Output: False

---

and syntax:

operand1 and operand2

Return true if its both operands are true.

e.g.

```
b = (6<10 and 10>8)
print(b)
```

Output: True

---

any() syntax:

any()

Return true if any element in a collection is true

e.g.

```
mylist = [True, False, True]
print(any(mylist))
```

Output: True

---

append text syntax:

open(“fileName”, “a”)

write(“text”)

Open a file by using “a” mode for appending text

write(“text”) writes text to the file

e.g.

```
f = open("myFile.txt", "a")
f.write(" This is the appended text.")
```

Output: Please check myFile.txt, we can fine the text "This is the appended text" is appended to the file.

---

append() syntax:

append(element)

```
# Append an element at the end of the list
e.g.
animals = ["ass", "bat", "cow"]
animals.append("dog")
print(animals)
# Output:  ['ass', 'bat', 'cow', 'dog']
```

---

appendleft() syntax:

dequeObject.appendleft(item)

```
# Append an item from left end of the deque
e.g.
from collections import deque
dq = deque([1,2,3,4,5])
dq.appendleft(0)
print(dq)
# Output:   deque([0, 1, 2, 3, 4, 5])
```

---

as syntax:

import module as alias

```
# Create an alias of the module
e.g.
import calendar as c
print(c.month_name[8])
# Output:  August
```

---

ascii() syntax:

ascii()

```
# Returns a readable text by escaping non-ascii characters
e.g.
text = ascii("This village is called Båtåky")
print(text)
# Output:  'This village is called B\xe5t\xe5ky'
```

---

assert() syntax:

assert (test-expression), error-message

In assert statement, if test expression returns False, an error message will appear.

e.g.

```
myList = ["a", "b", "c", "d", "e"]
size = len(myList)       # len(myList) returns 5
assert (size == 5), "The length of the list is abnormal"
print (size)   # if the size is not 5, error message will be shown
```

Output: 5

Explanation:

"**assert (size == 5),** "The length of the list is abnormal"" is an assertion statement, which executes the (size == 5) first, if it returns false, the error message "The length of the list is abnormal" will appear.

---

bin() syntax:

bin()

Return a binary number starting with 0b.

e.g.

```
num = bin(9)
print(num)
```

Output: 0b1001

---

bool() syntax:

bool(value)

Check if the value is true or false

e.g.

```
print(bool(0))
print(bool(100))
print(bool(None))
```

Output: False, True, False

---

break syntax:

break

Stop running from a loop according to the condition.

e.g.

```
for n in range(6):
```

```
    if n == 3:
        break
    print(n)
# Output:    0   1   2
```

---

bytearray() syntax:

bytearray()

```
# Return an array with a specified byte length.
e.g.
n= bytearray(2)      # set 2 bytes' length
print(n)
# Output:  bytearray(b'\x00\x00)
```

---

bytes() syntax:

bytes()

```
# Return a bytes object
e.g.
n = bytes(6)
print(n)
# Output:  b'\x00\x00\x00\x00\x00\x00'
```

---

calendar syntax:

import calendar

cal = calendar.month(year, month)

```
# Get a calendar of the year and month
e.g.
import calendar
cal = calendar.month(2023, 3)
print(cal)
# Output:   Return a calendar in Mar, 2023
```

---

callable() syntax:

callable()

```
# Return true if a function is callable
e.g.
def func():
```

```
    print(10)
print(callable(func))
# Output:  True
```

---

capitalize() syntax:

capitalize()

```
# Change the first letter to uppercase
e.g.
str = "jQuery"
print(str.capitalize())
# Output:  Jquery
```

---

casefold() syntax:

string.casefold()

```
# convert a string to lower case
e.g.
str = "JQuery in 8 Hours"
s = str.casefold()
print(s)
# Output:  jquery in 8 hours
```

---

casting syntax:

int(), str(), float()

```
# convert types to int, str, float respectively
e.g.
print(str(8))    # converted to a string
print(int(8))    # converted to an integer
print(float(8))    # converted to a float number
# Output:    8    8    8.0
```

---

ceil() syntax:

math.ceil( );

```
# Return an integer that is greater than or equal to its argument.
e.g.
import math
print(math.ceil(9.5))
```

```
# Output:  10
```

---

center() syntax:

center(w, f)

```
# Center the string with width w and fill with f
e.g.
str = "this is a center example"
print(str.center(35, '$'))
# Output:  $$$$$$this is a center example$$$$$
```

---

chain() syntax:

chain(iterable1, iterable2, iterable3,…)

```
# Connect multiple iterables together and return a single iterable
e.g.
from itertools import chain
odd = [11, 13, 15, 17, 19]
even = [12, 14, 16, 18, 20]
num = list(chain(odd, even))
print(num)
# Output:   [11, 13, 15, 17, 19, 12, 14, 16, 18, 20]
```

---

character syntax:

string[index]

```
 # Return a character of the string at the specified index
e.g.
str = "Python"
print(str[2])
# Output: t
```

---

character syntax:

string[index1: index2]

```
 # Return characters from index1 to index2-1
e.g.
myString = "Python is a good language"
print(myString[7:16])
# Output:   is a good
```

---

chdir() syntax:

os.chdir(path)

Change a directory

e.g.

```
os.chdir("c:\\mydir")    # Set “mydir” as a current working directory
cwd = os.getcwd()
print("The current working directory is:", cwd)
# Output:  The current working directory is: c:\\mydir
```

---

chr() syntax:

chr()

Return a character from a Unicode code

e.g.

```
c = chr(86)
print(c)
# Output:  V
```

---

class syntax:

```
class ClassName:       # define a class
    classVariable = value          # declare a class variable
    def  __init__(self):       #  declare a constructor,
    def classMethod(self):      # define a class method
```

e.g.

```
class Animal:      # define a class Animal
    count = 88      # declare a variable
    def __init__(self, value1, value2):   # define a constructor
        self.name = value1     # initialize the variable
        self.age = value2     # "self" is the current object
    def show(self):      # define a method
        print ("The animal name is " + self.name)
        print ("The tiger age is "+ self.age)
tiger = Animal("Tiger", "100")    # create an object
tiger.show()      # object references method
print ("Tiger counts " + str(tiger.count))  # object references variable
# Output:
```

```
The animal name is Tiger
The tiger age is 100
Tiger counts 88
```

---

class BaseClass syntax:

```
class BaseClass:     # define a base class
……
class DerivedClass (BaseClass):     # define a derived class
……
# The derived class inherits all members of the base class.
e.g.
class Computer:     # define a base class
    harddrive = 10000
    memory = 8
    def setValue(self, harddrive, memory):   # base method
        Computer.harddrive = harddrive
        Computer.memory = memory
class Desktop(Computer):     # define a derived class
    def capacity(self):       # derived method
        print ("Harddrive capacity: " + str(self.harddrive))
        print ("Memory capacity: " + str(self.memory))
D = Desktop()     # create an object "D"
D.setValue( 9000, 7 )  # call the base method "setValue(){ }"
D.capacity()      # call the derived method "capacity(){ }"
# Output:
Harddrive capacity: 9000
Memory capacity: 7
```

---

clear() syntax:

```
clear()
# Clear all elements of a collection
e.g.
animals = ["ass", "bat", "cow"]
animals.clear()
print(animals)
# Output:  [ ]
```

---

close() syntax:

file.close()

Close a file

e.g.

```
f = open("myfile.txt", "r")
print("After opening a file, we close it now.")
f.close()
# Output:  After opening a file, we close it now.
```

---

command prompt syntax:

>>>

This symbol is the Python interactive command shell prompt, requests the input from the user.

e.g.

```
>>> " Hello " + " World! "
# Output:  Hello World!
```

---

comment syntax:

#

This symbol is used as a comment symbol.

e.g.

```
print("Hello World! ")     # print() is used to output contents
# Output:  Hello World!
```

---

compile() syntax:

import re # import an regular expression module

re.compile(regular expression)

Return a pattern object by compiling the regular expression.

e.g.

```
import re       # import re module
pattern = re.compile("^(\d{3})-(\d{3})-(\d{4})$")    # return a pattern
phoneNumber = input("Enter your phone number:")
```

```
valid = pattern.match(phoneNumber)      # match
print (phoneNumber)
if valid:
    print ("Valid Phone Number!")
# Output:
Enter your phone number:  123-123-1234
Valid Phone Number!
```

---

compile() syntax:

compile('source','filename', 'mode')
exec(compiled_code)

```
# Return a code by compiling a source object in a file and using exec() to execute the compiled code.
# The "mode" parameter is as follows:
eval : used if the source is an expression only
exec : used if the source is a block of statements
single : used if the source is a single interactive statement
e.g.
source = 'print(168)'
code = compile(source, 'myfile', 'exec')
exec(code)
# Output:  168
```

---

complex() syntax:

complex(x, y)

```
# Convert the real number x and imaginary number y into a complex number
e.g.
n = complex(6, 8)
print(n)
# Output:  (6+8j)
```

---

compress() syntax:

itertools.compress(list, selector)

```
# Return items of an iterator corresponding to a given list.
e.g.
```

```
import itertools
import operator
fruit =['apple', 'banana', 'cherry', 'date']
select = [False, False, True, False]
favorite = itertools.compress(fruit, select)
for item in favorite:
    print("My favorite fruit is: ", item)
# Output:  My favorite fruit is:  cherry
```

---

connect lists syntax:

list1 + list2

```
# Connect two lists
e.g.
lst1 = [0, 1, 2]
lst2 = [3, 4, 5]
print(lst1 + lst2)
# Output:  [0, 1, 2, 3, 4, 5]
```

---

connect string syntax:

```
+
# Concatenate two strings together
e.g.
myString = "Python "+ "is a good language"
print(myString)
# Output:  Python is a good language
```

---

continue syntax:

```
continue
# Skip the next command and continue the next loop.
e.g.
for n in range(6):
    if n == 3:
        continue
    print(n)
```

```
# Output:  0 1 2 4 5     (Note: the output has no "3")
```

---

convert data type syntax:

type(data)

```
# Convert the data to a specified type, such as str(data), int(data)……
e.g.
 str("Golang in 8 Hours")    # convert to str type
 int(10.68)    # convert to int type
 float(10)     # convert to float type
 complex(1j)    # convert to complex
 list(("red", "yellow", "green"))     # convert to list type
 tuple(("red", "yellow", "green"))     # convert to tuple type
 range(8)    # convert to range type
 dict(name="Smith", age=18)          # convert to dict type
 set(("red", "yellow", "green"))    # convert to set type
 frozenset(("red", "yellow", "green"))          # convert to frozenset
 bool(10)     # convert to bool type
 bytes(10)    # convert to bytes type
 bytearray(10)             # convert to bytearray
type

 memoryview(bytes(10))     # convert to memoryview type
```

---

copy() syntax:

collection.copy()

```
# Return a copy of a collection
e.g.
animals = ["ass", "bat", "cow"]
animals.copy()
print(animals)
# Output:  ['ass', 'bat', 'cow']
```

---

count() syntax:

count(ch)

```
# Count the number of the specified character
```

e.g.

```
str = "jQuery is a great language!"
print(str.count("a"))
# Output: 4
```

---

count() syntax:

collection.count(value)

```
# Count the number a specified value in a collection
e.g.
myTuple = (3, 8, 6, 5, 9, 2, 7, 6, 1, 6, 8)
print(myTuple.count(6))
# Output:  3
```

---

ctime() syntax:

time.ctime()

```
# Convert a time in seconds since the epoch to a string in local time.
e.g.
import time
print(time.ctime())
# Output:   Wed Mar 29 18:32:46 2023
```

---

cycle() syntax:

itertools.cycle(iterable)

```
# Create an iterator, which contains all of the iterable elements.
e.g.
from itertools import cycle
ch=0
for item in cycle('abc'):
    print(item, end='  ')
    ch+=1
    if(ch==10):     # only print 10 characters
        break
# Output:   a  b  c  a  b  c  a  b  c  a
```

---

date() syntax:

```
dt = datetime.datetime.today()     # create a datetime object
```

d = dt.date() # get the date object from datetime object
Return the current date and time.
e.g.
```
import datetime
dt = datetime.datetime.today()     # create a datetime object
d = dt.date()     # get the date object from datetime object
print(dt)
print(d)
```
Output:
2023-03-29 15:27:20.682288
2023-03-29

---

datetime() syntax:

dateObj = datetime.datetime(year, month, day)
Create a date object
e.g.
```
import datetime
dt = datetime.datetime(2023, 3, 28)
print(dt)
```
Output: 2023-03-28 00:00:00

---

decode() syntax:

dict = demjson.decode(json)
Decode a Json string, return a Python dictionary
Demjson is a third-party module for encoding python or decoding json
Before using "demjson", we may download & install demjson module.
e.g.
```
import demjson
dict = demjson.decode(json)
```
Return a python dictionary
To know more "demjson", please reference the related website.

---

def function( ) syntax:

```
def functionName( ):      # define a function
    function body
……
```

functionName( ) # call a function

e.g.

```
def myFunction( ):
    print("This is a custom function.")
myFunction( )
# Output: This is a custom function.
```

---

def function(arguments) syntax:

def functionName(arguments):

function body

……

functionName(arg)

```
# Define a function with arguments
# Call a function with an argument
e.g.
def userName(name):
    print( "My name is " + name)
userName("Andy")
# Output: My name is Andy
```

---

del syntax:

del Class/object

```
# Delete a class, object, variable, list, set, or tuple……
e.g.
mylist = ["apple", "banana", "cherry"]
del mylist[1]
print(mylist)
# Output: ['apple', 'cherry']
```

---

delattr() syntax:

delattr(class, attribute)

```
# Delete an attribute from a class
e.g.
```

```
class Student:
    name = "Andy"
    id = "0026"
delattr(Student, 'id')
# Delete the attribute 'id' from the class 'Student'
```

---

def main()syntax:

```
def main():
      function body
# The main( ) function is a default start point of the whole program.
e.g.
def main():
     print("Very good!")
main()      # call main()
# Output:   Very good!
```

---

deque() syntax:

```
obj = deque(['item1', 'item2', 'item3', …])
# A deque is a double-ended queue in which elements can be inserted and removed from both end of the queue.
e.g.
from collections import deque
dq = deque(['Name', 'Id', 'Age'])
print(dq)
# Output:   deque(['Name', 'Id', 'Age'])
```

---

dict() syntax:

```
dict()
# Create a dictionary
e.g.
d = dict(name = "Andy", id = "0026", age = 18)
print(d)
# Output:  {'name': 'Andy', 'id': '0026', 'age': 18}
```

---

dictionary syntax:

```
dictionaryName = { key1: val1,  key2:val2,  key3:val3 }
```

```
# Define a dictionary
e.g.
light = {0:"red", 1:"yellow", 2:"green"}   # create a dictionary
print(light)      # show all keys:values in “light”
# Output:  {0: 'red', 1: 'yellow', 2: 'green'}
```

---

dictionary clear() syntax:

```
dictionary.clear()
# Clear all elements in a dictionary
e.g.
dict = {1: "apple", 2: "banana", 3: "cherry"}
dict.clear()
print("dict = ", dict)
# Output:   dict = { }
```

---

dictionary copy() syntax:

```
dict2 = dict1.copy()
# Copy all elements of dict1 to dict2
e.g.
dict1 = {1: 'apple', 2: 'banana', 3: 'cherry'}
dict2 = dict1.copy()
print('dict2 = ', dict2)
# Output:   dict2 =  {1: 'apple', 2: 'banana', 3: 'cherry'}
```

---

dictionary fromkeys() syntax:

```
dict.fromkeys(key, value)
# Create a dictionary containing the keys and values
e.g.
k = ('key1', 'key2', 'key3', 'key4')
v = 10
myDict = dict.fromkeys(k, v)
print(myDict)
# Output:  {'key1': 10, 'key2': 10, 'key3': 10, 'key4': 10}
```

---

dictionary get() syntax:

```
dict.get(key)
```

```
# Return the value according to the specified key
e.g.
myDict = {0:"apple", 1:"banana", 2: "cherry"}
print(myDict.get(1))
# Output:  banana
```

---

dictionary items() syntax:

```
dict.items()
# Return all items (key, value pairs) in the dictionary.
e.g.
myDict = {0:"apple", 1:"banana", 2: "cherry"}
print(myDict.items())
# Output:  dict_items([(0, 'apple'), (1, 'banana'), (2, 'cherry')])
```

---

dictionary keys() syntax:

```
dictionary.keys()
# Return a list containing all keys in the dictionary
e.g.
dict = {'ant':1, 'bee': 2, 'cat': 3}
print(dict.keys())
# Output:  dict_keys(['ant', 'bee', 'cat'])
```

---

dictionary pop() syntax:

```
dict.pop(key)
# Remove an item according to the specified key.
e.g.
myDict = {0:"apple", 1:"banana", 2: "cherry"}
removed_item = myDict.pop(1)
print(removed_item)
# Output:  banana
```

---

dictionary popitem() syntax:

```
dict.popitem()
# Remove the last item of a dictionary
e.g.
student = {"name": "Andy", "age": "17", "id": "0026"}
```

print("The removed item is:", **student.popitem()**)
# Output: The removed item is: ('id', '0026')

---

dictionary setdefault() syntax:

dict.setdefault(key, default_value)
Return the item's value if the key is in the dictionary.
Return the default value if the key is not in the dictionary.
e.g.
dict = {'name': 'Andy', 'id': '0026'}
print ("id =", **dict.setdefault('id', None)**)
print ("age =", **dict.setdefault('age', None)**)
Output: id = 0026 age = None

---

dictionary update() syntax:

dict1.update(dict2)
Update the items of dict1 by using dict2
e.g.
dict1 = {0:"apple", 1:"banana", 2: "cherry"}
dict2 = {1:"berry", 2:"cashew"}
dict1.update(dict2)
print("dict1 = ", dict1)
# Output: dict1 = {0: 'apple', 1: 'berry', 2: 'cashew'}

---

dictionary values() syntax:

dictionary.values()
Return a list containing all values in the dictionary
e.g.
dict = {'ant':1, 'bee': 2, 'cat': 3}
print (dict.values())
Output: dict_values([1, 2, 3])

---

difference() syntax:

set1.difference(set2)
Return the difference between two sets
e.g.
set1 = {"ant", "bee", "cat"}

```
set2 = {"ant", "bee", "cow"}
diff = set1.difference(set2)
print(diff)
# Output:  {'cat'}
```

---

difference_update() syntax:

set1.difference_update(set2)

```
# Remove the same elements existing in both sets
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ant", "bee", "cow"}
set1.difference_update(set2)
print("Now the set1 is:")
print(set1)
# Output:  Now the set1 is: {'cat'}
```

---

dir() syntax:

dir(object)

```
# Display all functions or properties of the specified object.
e.g.
import math
print(dir(math))     # display all attributes and methods of "math"
# Output:    ['__doc__', '__loader__', '__name__', '__package__',.....
```

---

discard() syntax:

set.discard('element')

```
# Remove the specified element from a set
e.g.
mySet = {"ant", "bee", "cat"}
mySet.discard("bee")
print(mySet)
# Output:  {'ant', 'cat'}
```

---

divmod() syntax:

divmod(x, y)

```
# Return a quotient and a remainder of x divided by y
```

e.g.
```
n = divmod(11, 3)
print(n)
# Output: (3,2)
```
---

double quotes syntax:

str = "string"
Define a string using a pair of double quotes
e.g.
```
str = "I am a string enclosed by a pair of double quotes"
print(str)
# Output: I am a string enclosed by a pair of double quotes
```
---

dumps() syntax:

import json
json.dumps()
Convert python to json, return a json string
e.g.
```
import json
p = { "name": "Andy", "age": 18, "id": "007" }     # python dictionary
j = json.dumps(p)     # return a json string
print(j)
# Output: {"name": "Andy", "age": 18, "id": "007"}
```
---

elements() syntax:

counterObj.elements()
Return all elements of the Counter class in ascending order
e.g.
```
from collections import Counter     # import Counter class
obj = Counter("cheatsheet")      # create an object
for item in obj.elements():
    print (item)    # return all elements in ascending order
# Output: c h h e e e a t t s
```
---

elif syntax:

elif condition

```
# Same as 'else if'
e.g.
n = 2
if n == 1:
    print("1")
elif n == 2:
    print("2")
else:
    print("3")
# Output:  2
```

---

else syntax:

```
else:
# Otherwise
e.g.
n = 0
if n == 1:
    print("1")
else:
    print("0")
# Output:  0
```

---

encode() syntax:

```
json = demjson.encode(dict)
# Encode a Python dictionary, return a Json string
# Demjson is a third-party module for encoding python or decoding json
# Before using "demjson", we may download & install demjson module.
e.g.
import demjson
json = demjson.encode(dict)
# Return a json string
# To know more "demjson", please reference the related website.
```

---

endwith() syntax:

```
endwith(param)
# Check if the string ends with the param.
```

e.g.
```
str = "Perl in 8 Hours!"
s = str.endswith("!")
print(s)
# Output:  True
```
---

enumerate() syntax:

enumerate(collection)
Convert a collection to an enumerate object
e.g.
```
m = ('ant', 'bee', 'cat')
n = enumerate(m)
print(list(n))
# Output:  [(0, 'ant'), (1, 'bee'), (2, 'cat')]
```
---

environment variable syntax:

set PYTHONPATH=path
Set an environment variable in Unix
e.g.
```
set PYTHONPATH=/usr/local/lib/python
# Note: The format of Unix path is:   /…/…/…
```
---

environment variable syntax:

set PYTHONPATH=path;
Set an environment variable in Windows
e.g.
```
set PYTHONPATH=c:\python30\lib;
# Note: The format of Windows path is:   \...\...\...
```
---

evel() syntax:

evel(expression)
Evaluate the expression
e.g.
```
n = 'print(68)'
eval(n)
# Output:  68
```

---

except syntax:

except:

Used in exceptions

e.g.

```
try:
    value = 100/0
except:
    print("Error Occurs!")
# Output:  Error Occurs!
```

---

exception syntax:

try:

……

except XxxError as message:

……

“try block” contains the code that may cause an exception.

“except block” catches the error, and handles the exception.

e.g.

```
try:
    value = 100/0
except ValueError as message:
    print("Exception occurs!", message)
# Output:  Traceback (most recent call last):….value = 100/0
                ZeroDivisionError: division by zero
```

---

exec() syntax:

exec(source_code)

Execute the specified source code

e.g.

```
sc = 'name = "Ray Yao"\nprint(name)'
exec(sc)
# Output:  Ray Yao
```

---

expandtabs() syntax:

string.expandtabs()

```
# Expand the tabs some whitespaces
e.g.
str = "J\tA\tV\tA"
s =  str.expandtabs(3)      # three whitespaces
print(s)
# Output:  J   A   V   A
```

---

extend() syntax:

list1.extend(list2)

```
# Extend one list with another list
e.g.
list1 = ["ass", "bat", "cow"]
list2 = ["ant", "bee", "cat"]
list1.extend(list2)
print(list1)
# Output:  ['ass', 'bat', 'cow', 'ant', 'bee', 'cat']
```

---

file.close() syntax:

fileObj.close()

```
# Close a file
e.g.
myfile.close()
```

---

fileno() syntax:

fileObj.fileno()

```
# Return a number that represents the file descriptor of the stream.
e.g.
f = open("myfile.txt", "r")
print(f.fileno())
# Output:  3
```

---

filter() syntax:

filter(function, collection)

```
# Filter the numbers in the collection by running the function.
e.g.
collection = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
def func(number):
    if number % 2 == 0:
        return True
    return False
evenNum = list(filter(func, collection))    # convert to list
print(evenNum)
# Output:  [2,4,6,8,10]
```

---

finally syntax:

finally:

In “try/except” block, “finally” statement is the code that must be executed.

e.g.

```
try:
    value = 100/0
except:
    print("Error Occurs!")
finally:
    print("Remind: Please be careful when you input data!")
# Output:    Error Occurs!
             Remind: Please be careful when you input data!
```

---

find() syntax:

find(c)

Return the index of the first occurrence, or -1

e.g.

```
str = "JavaScript"
print(str.find("v"))
# Output:  2
```

---

findall() syntax:

findall(string[, pos, endpos)

Find all the substrings that the regular expression matches in the string and returns a list

pos: start position

endpos: ending position

e.g.

```
import re
pattern = re.compile(r'\d+')    # only find digital number
list = pattern.findall('ok23vary168good333', 0, 30)
print(list)
# Output:  ['23', '168', '333']
```

---

finditer() syntax:

import re

re.finditer(pattern, string)

Find all the substrings that the regular expression matches in the string, and returns an iterator

e.g.

```
import re
iter = re.finditer(r"\d+","10a30bc50jf90")
for match in iter:
    print(match.group())
# Output:  10  30  50  90
```

---

float() syntax:

float(number)

Convert an integer number to a floating point number.

e.g.

```
n = float(10)
print(n)
# Output:  10.0
```

---

floor() syntax:

math.floor( );

Return an integer that is less than or equal to its argument.

e.g.

```
import math
print(math.floor(9.5))
# Output:  9
```

---

flush() syntax:

file.flush()
Clear the buffer when writing a file
e.g.
```
f = open("myfile.txt", "a")
f.write("Hello World!")
f.flush()
print("The buffer has been cleared!")
# Output:  The buffer has been cleared!
```
---

for syntax:

for <variable> in <sequence> :
<statements>
Repeats a given code block by the specified number of times.
e.g.
```
for str in 'Good':
    print(str)
# Output:  G o o d
```
---

for…in tuple syntax:

for elements in tuple
Iterate through the elements of a tuple
e.g.
```
myTuple = ("apple", "banana", "cherry")
for elements in myTuple:
    print(elements)
# Output:  apple  banana  cherry
```
---

for var in range(n1, n2) syntax:

for var in range(n1, n2)
Generate a sequence from n1 to n2-1.
e.g.
```
for num in range(3,10) :
    print(num)
# Output:     3,4,5,6,7,8,9
```
---

format() syntax:

format(value, type)
Format a specified value
e.g.
```
n = format(0.85, '%')
print(n)
```
Output: 85.000000%

---

format() syntax:

string.format(number)
format() can format a number, and put the number in {} to combine a string.
e.g.
```
n = 100
s = "My score on the test is {}"
print(s.format(n))
```
Output: My score on the test is 100

---

format printing syntax:

print(" %s %d %c %f %h %o" % ('string', digit, ascii, float, hex, oct))
Output a formatted string
e.g.
```
print("My name is %s and age is %d" % ('Ray', 38))
```
Output: My name is Ray and age is 38

---

from file import* syntax:

from file import*
Import anything from another file.
e.g.
```
from support import *
```
Import anything from the support file

---

from module import * syntax:

from module_name import *
Imports any members from a specified module.

e.g.

```
from support import *
# Import anything from the "support" module to the current file.
```

---

fromkeys() syntax:

dict.fromkeys(key, value)

```
# Create a dictionary containing the keys and value
e.g.
k = ('key1', 'key2', 'key3', 'key4')
v = 10
myDict = dict.fromkeys(k, v)
print(myDict)
# Output: {'key1': 10, 'key2': 10, 'key3': 10, 'key4': 10}
```

---

frozenset() syntax:

frozenset(collection)

```
# Freeze the collection, make it immutable.
e.g.
mylist = ['ant', 'bee', 'cat']
print(frozenset(mylist))
# Output: frozenset({'ant', 'bee', 'cat'})
```

---

function syntax:

```
def functionName( ):
    function body
……
functionName( )
# Define a function
# Call a function
e.g.
def myFunction( ):
    print("This is a custom function.")
myFunction( )
# Output: This is a custom function.
```

---

function with argument syntax:

def functionName(arguments):

    function body

……

functionName(arg)

Define a function with arguments

Call a function with an argument

e.g.

```
def userName(name):
    print( "My name is " + name)
userName("Andy")
# Output:  My name is Andy
```

---

get() syntax:

get("key")

Return a value of the specified key in the dictionary

e.g.

```
student = {
    "name": "Andy",
    "id": "0026"
}
print(student.get("id"))
# Output:  0026
```

---

getattr() syntax:

getattr(object)

get the attribute of the specified object

e.g.

```
class Student:
    name = "Andy"
    id = "0026"
print(getattr(Student, 'id'))
# Output:  0026
```

---

getcwd() syntax:

path = os.getcwd()
Return the current working directory
e.g.
os.chdir("c:\\mydir") # Set “mydir” as a current working directory
cwd = os.getcwd()
print("The current working directory is:", cwd)
# Output: The current working directory is: c:\\mydir
---

global syntax:

global variable
To define a global variable inside a function, we need to use "global" keyword. The global variable can be accessed everywhere.
e.g.
```
def fun():
    global myVar    # myVar becomes a global variable now
    myVar = "Cheat Sheet"
fun()
print("Html Css " + myVar)
```
Output: Html Css Cheat Sheet
---

globals() function syntax

globals()
The globals() function returns a dictionary containing the variables defined in the global namespace.
e.g.
print(globals())
# Output: {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': ……}
---

hasattr() syntax

hasattr(object, attribute)
Return true if the object has the specified attribute
e.g.
```
class Student:
    name = "Andy"
    id = "0026"
```

```
print(hasattr(Student, 'id'))
# Output: True
```

---

hash() syntax:

hash(object)

```
# Return a hash value of the specified object
e.g.
str = 'Scala in 8 Hours'
print(hash(str))
# Output: 160379786619873168
```

---

help() syntax:

help(object)

```
# Reference the built-in Python help system about the object
e.g.
help(list)
# Output:
Help on class list in module builtins:
class list(object)
|  list(iterable=(), /)
|  Built-in mutable sequence.
.....
```

---

header syntax:

HTTP fieldName: field contents

```
# Specify the format of http header
e.g.
Content-type: text/html
# The "content-type :text/ html" is the part of the HTTP header that tells the browser the Content type of the file.
```

---

hex() syntax:

hex(number)

```
# Converts a number to a hexadecimal value
e.g.
print(hex(168))
```

Output: 0xa8

---

http syntax:

HTTP fieldName: field contents

Specify the format of http header

e.g.

Content-type: text/html

The "content-type :text/ html" is the part of the HTTP header that tells the browser the Content type of the file.

---

id() syntax:

id(object)

Return an id of the specified object or return a random memory address of the specified object

e.g.

```
str = "Pandas in 8 Hours"
print(id(str))
```

Output: 139680528469008

This result will be different every time the program runs

---

if syntax:

```
if  test-expression:
    statements
```

Execute the statement only if a specified condition is true, does not execute any statement if the condition is false.

e.g.

```
x=100
y=200
if y > x:
    print("y is greater than x.")
```

Output: y is greater than x

---

if syntax:

```
if  test-expression:
    statements     # run when test-expression returns true
else:
```

```
    statements     # run when test-expression returns false
e.g.
x=100
y=200
if x > y:
    print("x is greater than y.")
else:
    print("x is less than y")
# Output:  x is less than y
```

---

if-elif syntax:

```
if condition1:
    run statement1……
elif condition2:
    run statement2……
elif condition3:
    run statement3……
else:
    run statement4……
# If a condition is satisfied, its statement will run, otherwise next elif runs.
e.g.
num=100
if num == 59:
    print ('fail')
elif num == 60:
    print ('pass')
elif num == 100:
    print ('excellent')
elif num < 0:
    print ('error')
else:
    print ('nothing')
# Output:  excellent
```

---

if / else syntax:

(if-true-do-this) if (test-expression) else (if-false-do-this)

If true does the first part. If false does the last part.
e.g.
```
a=100
b=200
result = "apple" if (a<b) else "banana"
print(result)
```
Output: apple

---

import file syntax:

from file import*
Import anything from another file
e.g.
```
from support import *
```
Import anything from the support file

---

import calendar syntax:

import calendar
cal = calendar.month(year, month)
Get a calendar of the year and month
e.g.
```
import calendar
cal = calendar.month(2020, 1)
print(cal)
```
Output: (Return a calendar on Jan. 2020)

---

import module syntax:

import module
Import an external module into the current file.
e.g.
```
import support      # import support module
support.myfunction()
```
Call myfunction in the support module.

---

import os syntax:

import os
Import os module for directory or file.

e.g.

```
import os
os.mkdir("/tmp/home/mydir”)
# You can use getcwd() to check it.
```

---

import re syntax:

import re

```
# Import re module for regular expression
```

e.g.

```
import re
pattern = re.compile("^(\d{3})-(\d{3})-(\d{4})$")
# returns a telephone number format of USA
```

---

import smtplib syntax:

import smtplib

```
# Import smtplib module for email
```

e.g.

```
import smtplib
obj = smtplib.SMTP('localhost')
obj.sendmail(sender, receivers, message.as_string())
# obj.sendmail() sends email
```

---

import time syntax:

import time

```
# Import time module for date and time
```

e.g.

```
import time
local_time = time.asctime( time.localtime(time.time()) )
print ("The local time is :", local_time)
# Output:   The local time is : Tue Mar 28 15:06:02 2023
```

---

import webbrowser syntax:

import webbrowser

```
# import webbrowser module for url or web
```

e.g.

```
import webbrowser
url = "http://www.amazon.com"
webbrowser.open(url)     # Open a specified web page
# Output:   (Amazon web page)
```

---

in syntax:

in

```
# check a character existing in a string
e.g.
str = "C programing"
if ( "C" in str ):
     print ("C is in the string")
# Output:  C is in the string
```

---

index syntax:

string[index]

```
 # Return a character of the string at the specified index
e.g.
str = "Python"
print(str[2])
# Output:  t
```

---

index1 : index2 syntax:

string[index1: index2]

```
 # Return characters from index1 to index2-1
e.g.
myString = "Python is a good language"
print(myString[7:16])
# Output:   is a good
```

---

index() syntax:

index(character)

```
# Return the character index of the first occurrence, or alert error
e.g.
str = "abec"
print(str.index("e"))
```

```
# Output: 2
```

---

inherit syntax:

```
class BaseClass:   # define a base class
……
class DerivedClass (BaseClass):   # define a derived class
……
# The derived class inherits all members of the base class.
e.g.
class Computer:     # define a base class
    harddrive = 10000
    memory = 8
    def setValue(self, harddrive, memory):   # base method
        Computer.harddrive = harddrive
        Computer.memory = memory
class Desktop(Computer):     # define a derived class
    def capacity(self):       # derived method
        print ("Harddrive capacity: " + str(self.harddrive))
        print ("Memory capacity: " + str(self.memory))
D = Desktop()     # create an object "D"
D.setValue( 9000, 7 )  # call the base method "setValue(){ }"
D.capacity()      # call the derived method "capacity(){ }"
# Output:
Harddrive capacity: 9000
Memory capacity: 7
```

---

input syntax:

```
variable = input("prompt")
# Users need to input some text by keyboard.
e.g.
age = input("Please input your age: ")
print("Your age is: " + age )
# Output:   Please input your age: Your age is: 18
```

---

insert() syntax:

```
list.insert(position, element)
```

```
# Insert an element to the specified position of a list
e.g.
animals = ["ass", "bat", "cow"]
animals.insert(2,"buffalo")
print(animals)
# Output:['ass', 'bat', 'buffalo', 'cow']
```

---

int() syntax

int(number)

```
# Convert a floating point number to an integer number
e.g.
print(int(3.14))
# Output:  3
```

---

intersection() syntax:

set1.intersection(set2)

```
# Return the same elements both in two sets
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ant", "bee", "cow"}
same = set1.intersection(set2)
print(same)
# Output:  {'ant', 'bee'}
```

---

intersection_update() syntax:

set1.intersection_update(set2)

```
# Remove the different elements both in two sets
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ant", "bee", "cow"}
set1.intersection_update(set2)
print("Now the set1 is:")
print(set1)
# Output:  Now the set1 is: {'ant', 'bee'}
```

---

is syntax:

x is y

```
# Chect if two variables are equal
e.g.
x = 'Ruby in 8 Hours'
y = 'Perl in 8 Hours'
print(x is y)
# Output:  False
```

---

is_alive() syntax:

thread.is_alive()

```
# Return True if a  thread is alive, return False if not
e.g.
import time
import threading
def func():
    time.sleep(2)     # sleep 2 seconds
    print('myThread is running')
myThread = threading.Thread(target=func)
myThread.start()
print("Is myThread alive?", myThread.is_alive())
# Output:   Is myThread alive? True
                    myThread is running
```

---

isalnum() syntax:

isalnum()

```
# Return true if all characters are numbers or letters
e.g.
str = "year2018";
s = str.isalnum();
print(s)
# Output:   True
```

---

isalpha() syntax:

isalpha()

Return true if all characters are letters

e.g.

```
str = "year2018";
s = str.isalpha();
print(s)
 # Output:  False
```

---

isascii() syntax:

isascii()

Return true if all characters in a string are ascii characters

e.g.

```
str = "Hero007"
s = str.isascii()
print(s)
# Output:  True
```

---

isatty() syntax:

file.isatty()

Return True if the file stream is interactive or connected to a terminal device. Return False if not.

e.g.

```
f = open("myFile.txt", "r")
print(f.isatty())
# Output:  False
```

---

isdecimal() syntax:

isdecimal()

Return true if all characters are decimal numbers

u“number” defines a decimal number

e.g.

```
str = u"12345678";
s = str.isdecimal();
```

```
print(s)
# Output:  True
```

---

isdigit() syntax:

isdigit()

```
# Return true if all characters are digits
e.g.
str = "123456";
s = str.isdigit();
print(s)
# Output:  True
```

---

isdisjoint() syntax:

set1.isdisjoint(set2)

```
# Return true if any items in set1 are unrelated to any items in set2.
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cow"}
unrelated = set1.isdisjoint(set2)
print(unrelated)
# Output:  True
```

---

isidentifier() syntax:

isidentifier()

```
# Returns True if the string is a valid identifier
e.g.
str = "100percent"
s = str.isidentifier()
print(s)
# Output:  False
```

---

isinstance() syntax:

isinstance(object, instance)

```
# Return true if the object is the specified instance
```

e.g.
print(isinstance(3.14, float))
# Output: True
---

islower() syntax:

islower()

Return true if all characters are lowercase
e.g.
str = "abcde"
s = str.islower();
print(s)
# Output: True
---

isnumeric() syntax:

isnumeric()
Return true if the string is numeric.
e.g.
str = "168168"
s = str.isnumeric()
print(s)
# Output: True
---

isoformat() syntax:

now.isoformat() **utcnow.isoformat()**
Return date & time in the iso format (YYYY-MM-DD)
e.g.
from datetime import datetime
now = datetime.now() #create a now object
print(now.isoformat())
# Output: 2023-03-29T17:54:15.142166
---

isprintable() syntax:

isprintable()
Return true if the string is printable.

e.g.

```
str = "Scala in 8 Hours!"
s = str.isprintable()
print(s)
# Output:  True
```

---

isspace() syntax:

isspace()

Return true if the string is only whitespace

e.g.

```
str = "      "
s = str.isspace();
print(s)
# Output:   True
```

---

issubclass() syntax:

issubclass(subclass, class)

Return true if the one class is another class's subclass

e.g.

```
class Building:
    high = 100
class Room(Building):
    high = 5
print(issubclass(Room, Building))
# Output:  True
```

---

issubset() syntax:

set1.issubset(set2)

Return true if set1 is a subset of set2

e.g.

```
set1 = {"ant", "bee", "cat"}
set2 = {"ant", "bee", "cat", "dog", "ewe"}
ss = set1.issubset(set2)
print(ss)
# Output:  True
```

---

issuperset() syntax:

set1.issuperset(set2)

Return true if set1 is a super set of set2

e.g.

```
set1 = {"ant", "bee", "cat", "dog", "ewe"}
set2 = {"ant", "bee", "cat"}
ss = set1.issuperset(set2)
print(ss)
# Output:  True
```

---

istitle() syntax:

istitle()

Return true if the string is title-case string

e.g.

```
str = "We Like Python"
s = str.istitle();
print(s)
# Output:   True
```

---

isupper() syntax:

isupper()

Return true if all characters are uppercase

e.g.

```
str = "VERY GOOD"
s = str.isupper();
print(s)
# Output:   True
```

---

items() syntax:

dictionary.items()

Return the key-value pairs of a dictionary

e.g.

```
student = {
    "name": "Andy",
    "id": "0026"
}
print(student.items())
# Output:  dict_items([('name', 'Andy'), ('id', '0026')])
```

---

iter() syntax:

iter()

```
# Create an iterator object
e.g.
iterator = iter(["ant", "bee", "cat"])
print(next(iterator))
print(next(iterator))
print(next(iterator))
# Output:   ant  bee  cat
# next(iterator) returns one of the items in the collection.
```

---

join() syntax:

separator.join()

```
# Join the strings by separators
e.g.
separator = "-";
str = ("x", "y", "z");
print(separator.join( str ));
# Output:   x-y-z
```

---

join() syntax:

thread.join()

```
# The main thread stops running until the joined thread terminates
e.g.
from time import sleep
from threading import Thread
def fun():
    sleep(2)
    print('Joined thread finished running.')
```

```
thread = Thread(target=fun)      # create a new thread
thread.start()
print('Main thread is waiting for joined thread to terminate.')
thread.join()
print('Main thread resumes running after joined thread terminates.')
# Output:
Main thread is waiting for joined thread to terminate.
Joined thread finished running.
Main thread resumes running after joined thread terminates.
```

---

json.dumps() syntax:

```
import json
j = json.dumps(p)
# Convert python to json, return a json string
e.g.
import json
p = { "name": "Andy", "age": 18, "id": "007" }     # python dictionary
j = json.dumps(p)     # return a json string
print(j)
# Output:   {"name": "Andy", "age": 18, "id": "007"}
```

---

json.loads() syntax:

```
import json
p = json.loads(j)
# Convert from json to python, return a python dictionary.
e.g.
import json
j =  '{ "name":"Andy", "age":18, "id":"007"}'     # json string
p = json.loads(j)      # return a python dictionary
print(p)
# Output:   {'name': 'Andy', 'age': 18, 'id': '007'}
```

---

keys() syntax:

```
dictionary.keys()
# Return all keys of a dictionary
e.g.
```

```
student = {
    "name": "Andy",
    "id": "0026"
}
print(student.keys())
# Output:  dict_keys(['name', 'id'])
```

---

lambda syntax:

lambda arg1 arg2, arg3,… : expression

```
# Define a lambda, create an anonymous function
# A lambda function is an anonymous function actually
e.g.
sum = lambda num1, num2 : num1 + num2
# num1, num2 are parameters.
# num1+ num2 is an expression
e.g.
fun = lambda arg : arg + 8
print(fun(2))
# Output:   10
```

---

len() syntax:

len (param)

```
 # Return the length of the param
e.g.
myList = [3, 4, 5]
print(len(myList))
# Output:   3
```

---

link() syntax:

os.link(src, dst)

```
# Create a hard link from a source file to a destination file
e.g.
import os
src = 'myfile.txt'     # source file
dst = 'hisfile.txt'      # destination file
os.link(src, dst)
```

print("Create a hard link successfully**!**")
# Output**:** Create a hard link successfully**!**

---

list syntax:

list1 + list2

Connect two lists

e.g.

lst1 = [0, 1, 2]
lst2 = [3, 4, 5]
print(lst1 + lst2)
# Output**:** [0, 1, 2, 3, 4, 5]

---

list syntax:

list = ['string1', 'string2', 'string3']
Create a list

e.g.

mylist = ["a", "b", "c", "d"]
print(mylist)
# Output**:** ['a', 'b', 'c', 'd']

---

list append() syntax:

list.append('element')
Append an element to a list

e.g.

myList = ['ant', 'bee', 'cat']
myList.append('dog')
print(myList)
# Output**:** ['ant', 'bee', 'cat', 'dog']

---

list clear() syntax:

list.clear()
Clear all elements of a list

e.g.

myList = [1, 2, 3, 4, 5]

myList.clear()
print('myList = ', myList)
# Output: myList = [ ]
---

list copy() syntax:

list2 = list1.copy()
Copy all elements of list1 to list2
e.g.
list1 = [1, 2, 3, 4, 5]
list2 = list1.copy()
print('list2 = ', list2)
# Output: list2 = [1, 2, 3, 4, 5]
---

list count() syntax:

list.count('element')
Count how many a specified elements in a list
e.g.
myList = ['a', 'b', 'c', 'd', 'd', 'd']
count = **myList.count('d')** # count how many 'd'
print('The count of d is: ', count)
# Output: The count of d is: 3
---

list extend() syntax:

list1.extend(list2)
Extend list1 by using list2
e.g.
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list1.extend(list2)
print('After extending: ', list1)
# Output: After extending: [1, 2, 3, 4, 5, 6]
---

list index() syntax:

list.index('element')
Get the index of an element in a list
e.g.

```
myList = ['ant', 'bee', 'cat', 'dog']
index = myList.index('cat')
print("The index is: ", index)
# Output:  The index is:  2
```

---

list insert() syntax:

```
list.insert(index, 'element')
# Insert an element to a list at the specified index
e.g.
myList = ['a', 'b', 'd', 'e']
myList.insert(2, 'c')
print('myList = ', myList)
# Output:  myList =  ['a', 'b', 'c', 'd', 'e']
```

---

list pop() syntax:

```
myList.pop(index)
# Remove an element according to the specified index in a list, and return the removed element.
e.g.
myList = ['a', 'b', 'c', 'd']
removed_item = myList.pop(2)
print('Removed Element: ', removed_item)
# Output:  Removed Element:  c
```

---

list remove() syntax:

```
list.remove(element)
# Remove a specified element from a list
e.g.
myList = [1, 2, 3, 4, 5, 6]
myList.remove(4)    # remove '4'
print('myList = ', myList)
# Output:  myList =  [1, 2, 3, 5, 6]
```

---

list reverse() syntax:

```
list.reverse()
# Reverse all elements in a list
```

e.g.
myList = [1, 2, 3, 4, 5]
myList.reverse()
print('myList = ', myList)
# Output: myList = [5, 4, 3, 2, 1]
---

list sort() syntax:

list.sort(reverse=True/False)
Sort all elements of a list in ascending order if 'reverse=False'
Sort all elements of a list in descending order if 'reverse=True'
e.g.
myList = [6, 3, 7, 1, 5, 9, 2, 8]
myList.sort(reverse=True)
print(myList)
# Output: [9, 8, 7, 6, 5, 3, 2, 1]
---

list() syntax:

list(param)
Return a list
e.g.
myList = list("R in 8 Hours")
print(myList)
# Output: ['R', ' ', 'i', 'n', ' ', '8', ' ', 'H', 'o', 'u', 'r', 's']
---

listdir syntax:

os.listdir(path)
Display all directories and files in the path
e.g.
import os
path = "/"
contents = os.listdir(path)
print("The directories and files in '", path, "' :")
print(contents)
# Output: The directories and file in ' / ' :
['opt', 'run', 'tmp', 'lib64', 'dev', 'mnt', 'usr',……
---

ljust() syntax:

ljust(w,f)

Left adjust string with width w and fill with f

e.g.

```
str = "This is a left-adjust example:"
print(str.ljust(40, '$'))
# Output:  This is a left-adjust example: $$$$$$$$$$
```

---

json.loads() syntax:

import json

p = json.loads(j)

Convert from json to python, return a python dictionary.

e.g.

```
import json
j =  '{ "name":"Andy", "age":18, "id":"007"}'      # json string
p = json.loads(j)       # return a python dictionary
print(p)
# Output:  {'name': 'Andy', 'age': 18, 'id': '007'}
```

---

locals() syntax:

locals()

Returns a dictionary containing the local symbol table

e.g.

```
print(locals())
# Output:  {'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__',:……}
```

---

localtime syntax:

import time

local_time = time.asctime(time.localtime(time.time()))

Get local time

e.g.

```
import time
local_time = time.asctime( time.localtime(time.time()) )
print (local_time)
# Output:  Sun Oct 17 23:36:28 2020
```

---

lower() syntax:

lower()

Convert a string to lower case

e.g.

```
str = "JQuery in 8 Hours"
print(str.lower())
# Output:  jquery in 8 hours
```

---

lstrip() syntax:

lstrip()

Remove leading spaces of the string

e.g. print str.lstrip(' ');

```
str = "      This is a lstrip sample!     ";
print(str.lstrip( ))
# Output:  This is a lstrip sample!
```

---

main() syntax:

def main():

    function body

The main( ) function is a default start point of the whole Python program.

e.g.

```
def main():
    print("Very good!")
main()      # call main()
# Output:  Very good!
```

---

maketrans() syntax:

mytable = str.maketrans(x, y) # create a mapping table for a string

str.translate(mytable) # translate the mapping table

x, y are the characters in the string

e.g.

```
str = "Hello Hey!"
mytable = str.maketrans("e", "a")     # "a" replaces "e"
print(str.translate(mytable))
# Output:  Hallo Hay!
```

---

map() syntax:

list(map(function, param))

The param is a collection (list, tuple, set, dictionary).
Call the function and run each item of the collection, return a new list.
e.g.

```
def mul(n):
    return n*10
p = [1,2,3]
print(list(map(mul, p)))
# Output:  [10, 20, 30]
```

---

match() syntax:

import re

re.match(pattern, string)

Match a pattern from the beginning of the string, return an object if true. Otherwise, return noon.
span() returns the start index and ending index of the match.
Note: match() only makes a match from the beginning.
e.g.

```
import re
print(re.match('ray', 'ray@yahoo.com').span())
# Output:   (0, 3)
```

---

match() syntax:

pattern.match(string)

Match the pattern with string, return true or false
e.g.

```
import re       # import re module
pattern = re.compile("^(\d{3})-(\d{3})-(\d{4})$")    # return a pattern
phoneNumber = input("Enter your phone number:")
valid = pattern.match(phoneNumber)      # match
print (phoneNumber)
if valid:
    print ("Valid Phone Number!")
# Output:
```

Enter your phone number: 123-123-1234
Valid Phone Number!

---

math.acos() syntax:

math.acos(number)
Return the arc cosine of a number in radians
e.g.
```
import math
print(math.acos(0.5))
```
Output: 1.0471975511965979

---

math.asin() syntax:

math.asin(number)
Return the arc sine of a number in radians
e.g.
```
import math
print(math.asin(0.5))
```
Output: 0.5235987755982989

---

math.atan() syntax:

math.atan(number)
Return the arc tangent of a number in radians
e.g.
```
import math
print(math.atan(0.5))
```
Output: 0.4636476090008061

---

math.ceil() syntax:

math.ceil( );
Return an integer that is greater than or equal to its argument.
e.g.
```
import math
print(math.ceil(9.5))
```
Output: 10

---

math.copysign() syntax:

math.copysign(x, y)
Return the value of the x and the sign of y
e.g.
```
import math
parameter
print(math.copysign(10, -10))
print(math.copysign(-20, 20))
# Output:     -10.0     20.0
```
---

math.cos() syntax:

math.cos(number)
Return the cosine of a number in radians
e.g.
```
import math
print (math.cos(0.5))
# Output:   0.8775825618903728
```
---

math.degrees() syntax:

math.degrees(radian)
Convert a number from radian to degree
e.g.
```
import math
print (math.degrees(45))
# Output:   2578.3100780887044
```
---

math.e syntax:

math.e
Return the value of e, which is 2.718281828459045
e.g.
```
import math
print (math.e)
# Output:   2.718281828459045
```
---

math.exp() syntax:

math.exp(x)

Return E raised to the power of x (e^x).
e.g.
```
import math
print(math.exp(2))
print(math.exp(-3))
# Output:
7.38905609893065
0.049787068367863944
```
---

math.fabs() syntax:

math.fabs(number)
Return an absolute value of a number
e.g.
```
import math
print(math.fabs(-3.14))
print(math.fabs(-100))
# Output:    3.14    100
```
---

math.factorial() syntax:

math.factorial(number)
Return a factorial of a number
e.g.
```
import math
print(math.factorial(5))
print(math.factorial(6))
# Output:    120    720
```
---

math.floor( ) syntax:

math.floor( );
Return an integer that is less than or equal to its argument.
e.g.
```
import math
print(math.floor(9.5))
# Output:  9
```
---

math.fmod() syntax:

math.fmod(x/y)

```
# Return a remainder of x/y
e.g.
import math
print(math.fmod(10, 2))
print(math.fmod(10, 3))
print(math.fmod(10, 4))
# Output:     0.0     1.0     2.0
```

---

math.frexp() syntax:

math.frexp(number)

```
#Return mantissa and exponent of a number
e.g.
import math
print(math.frexp(3))
print(math.frexp(4))
# Output:       (0.75, 2)       (0.5, 3)
```

---

math.fsum() syntax:

math.fsum(iterable)

```
# Return the sum of all items in an iterable (tuple, list, etc.).
e.g.
import math
print(math.fsum([1, 2, 3]))     # print the sum of all items
print(math.fsum([1, 2, 3, 4]))
# Output:     6.0     10.0
```

---

math.gcd() syntax:

math.gcd(x,y)

```
# Get the greatest common divisor of x and y.
e.g.
import math
print (math.gcd(4, 6))
# Output:   2
```

---

math.inf syntax:

math.inf # return a positive infinity
-math.inf # return a negative infinity
e.g.

```
import math
print (math.inf)
print (-math.inf)
# Output:  inf  -inf
```

---

math.isclose() syntax:

math.isclose(number1, number2)
Return True if the values of two numbers are close to each other.
e.g.

```
import math
print(math.isclose(1.234, 1.2345))
print(math.isclose(1.234, 1.2340000001))
```

Output: False True

---

math.isfinite() syntax:

math.isfinite(number)
Return True if a number is finite, return False if not.
e.g.

```
import math
print(math.isfinite(100))
print(math.isfinite(math.inf))
print(math.isfinite(float("nan")))
# Output:  True  False  False
```

---

math.isinf() syntax:

math.isinf(number)
Return True if a number is infinite
e.g.

```
import math
print(math.isinf(100))
print(math.isinf(math.inf))
print(math.isinf(float("nan")))
```

```
# Output:     False     True     False
```

---

math.isnan() syntax:

math.isnan(value)

```
# Return True if a value is not a number.
e.g.
import math
print (math.isnan (100))
print (math.isnan (math.nan))
# Output:     False     True
```

---

math.isqrt() syntax:

math.isqrt(integer_number)

```
# Get the square root of the integer_number, and return a new integer
greater than and nearest the integer_number.
e.g.
import math
print (math.isqrt(10))
print (math.isqrt (17))
# Output:     3     4
```

---

math.lcm() syntax:

lcm = x/math.gcd(x,y)*y

```
# Get the least common multiple of x and y.
# Python has no lcm() actually, but we can use this equation to get lcm.
e.g.
import math
lcm = 3/math.gcd(3,4)*4
print(lcm)
# Output:   12.0
```

---

math.ldexp() syntax:

math.ldexp(x, y)

```
#Return value of x * (2**y)
e.g.
import math
```

```
print(math.ldexp(2, 3))
# Output:  16.0
```

---

math.log() syntax:

math.log(number)

```
# Return the natural logarithm of a number
e.g.
import math
print(math.log(3.14))
print(math.log(10))
# Output:  1.144222799920162  2.302585092994046
```

---

math.log2() syntax:

math.log2(number)

```
# Return the base-2 logarithm of a number
e.g.
import math
print(math.log2(3.14))
print(math.log2(2))
# Output:  1.6507645591169022  1.0
```

---

math.log10() syntax:

math.log10(number)

```
# Return the base-10 logarithm of a number
e.g.
import math
print(math.log10(3.14))
print(math.log10(10))
# Output:  0.49692964807321494  1.0
```

---

math.max( ) syntax:

math.max( );

```
# Return the greater one between two numbers.
e.g.
print(max(4,2))
# Output:  4
```

---

math.min( ) syntax:

math.min( );

Return the less number between two numbers.

e.g.

```
print(min(4,2))
# Output:  2
```

---

math.modf() syntax:

math.modf(number)

Return a value consisted of the fractional part and integer part.

e.g.

```
import math
print (math.modf(10))
print (math.modf(10.1))
print (math.modf(10.2))
# Output:
(0.0, 10.0)
(0.09999999999999964, 10.0)
(0.1999999999999993, 10.0)
```

---

math.nan syntax:

math.nan

Return a value of nan, instead of a number

e.g.

```
import math
print (math.nan)
# Output:  nan
```

---

math.pi syntax:

math.pi

Return the value of pi, which is 3.141592653589793

e.g.

```
import math
print (math.pi)
# Output:  3.141592653589793
```

---

math.pow( ) syntax:

math.pow( );

Return the first argument raised to the power of the second argument.

e.g.

```
import math
print (math.pow(4,2))
```

Output: 16.0

---

math.prod() syntax:

math.prod((num1, num2, num3…))

Return the product of a series of numbers

e.g.

```
import math
series = (1, 2, 3)
print(math.prod(series))
```

Output: 6

---

math.radians() syntax:

math.radians(degree)

Convert a number from degree to radian

e.g.

```
import math
print (math.radians(45))
```

Output: 0.7853981633974483

---

math.remainder() syntax:

math.remainder(x, y)

Return the remainder of x/y

e.g.

```
import math
print (math.remainder(21, 2))
print (math.remainder(21, 3))
```

Output: 1.0 0.0

---

math.sin() syntax:

math.sin(number)
Return the sine of a number in radians
e.g.
```
import math
print (math.sin(0.5))
# Output:  0.479425538604203
```
---

math.sqrt( ) syntax:

math.sqrt( );
Return the square root of the argument.
e.g.
```
import math
print (math.sqrt(4))
# Output:  2.0
```
---

math.tan() syntax:

math.tan(number)
Return the tangent of a number in radians
e.g.
```
import math
print (math.tan(0.5))
# Output:  0.5463024898437905
```
---

math.tau syntax:

math.tau
Return the value of tau, which is 6.283185307179586
e.g.
```
import math
print (math.tau)
# Output:  6.283185307179586
```
---

math.trunc() syntax:

math.trunc(number)
Return the truncated integer parts of at number
e.g.
```
import math
```

```
print(math.trunc(3.14))
print(math.trunc(-100.123))
# Output:     3     -100
```

---

max() syntax:

math.max( );

```
# Return the greater one between two numbers.
e.g.
print(max(4,2))
# Output:   4
```

---

memoryview() syntax:

memoryview(binary)

```
# Return a memoryview view of a binary object
e.g.
m = memoryview(b"1001")   # b"object" defines a binary object.
print(m)
# Output:  <memory at 0x7fdaddb7b580>
```

---

min() syntax:

math.min( );

```
# Return the less number between two numbers.
e.g.
print(min(4,2))
# Output:   2
```

---

mkdir() syntax:

os.mkdir()

```
# Create a new directory
e.g.
import os
os.mkdir("/tmp/home/mydir”)
print "A new directory is created"
# Output:   A new directory is created
```

---

module syntax:

from module import *

```
# Import anything from a module into the current file.
e.g.
from support import *
# import anything from support module
```

---

next() syntax:

next(item)

```
# Returns one item in an iterable collection
e.g.
iterator = iter(["ant", "bee", "cat"])
print(next(iterator))
print(next(iterator))
print(next(iterator))
# Output:  ant  bee  cat
```

None syntax:

None

```
# Represent a null value
e.g.
v = None
print(v)
# Output:  None
```

---

nonlocal syntax:

nonlocal variable

```
# Declare a non-local variable in a nested function
# The nonlocal variable does not belong to the inner function.
e.g.
def fun1():
    v = "Andy"
    def fun2():     # inner function
        nonlocal v     # declare a non local variable
        v = "Rose"    # v does not belong to the inner function
    fun2()
```

```
    return v
print(fun1())
# Output:  Rose
```

---

not syntax:

not operand

```
# Return true if the operand is false
e.g.
v = False
print(not v)
# Output:  True
```

---

not in syntax:

text not in string

```
# Check if the text is not in a string
e.g.
str = "Hello World!"
print("Hi" not in str)
# Output:  True
```

---

now() syntax:

datetime.datetime.now()

```
# Get the current day and time
e.g.
import datetime
dt = datetime.datetime.now()
print(dt)
# Output:   2023-03-29 15:21:17.903378
```

---

object creating syntax:

objectName = ClassName(args)

```
# Create an object
e.g.
class Animal:      # define a class Animal
    count = 88      # declare a variable
    def __init__(self, value1, value2):   # define a constructor
```

```
        self.name = value1     # initialize the variable
        self.age = value2     # "self" is the current object
    def show(self):      # define a method
        print ("The animal name is " + self.name)
        print ("The tiger age is "+ self.age)
tiger = Animal("Tiger", "100")     # create an object
tiger.show()      # object references method
print ("Tiger counts " + str(tiger.count))  # object references variable
# Output:
The animal name is Tiger
The tiger age is 100
Tiger counts 88
```

---

object() syntax:

```
object()
# Create a featureless object.
e.g.
myObj = object()
print(dir(myObj))
# Output:
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', ……]
```

---

oct() syntax:

```
oct(number)
# Convert the number to an octal value
e.g.
print(oct(18))
# Output:  0o22
```

---

open(“fileName”, “a”) syntax:

```
fileObj = open(“fileName”, “a”)
fileObj.write( “text” )
 # Open a file by using “a” mode for appending text
 # write( “text” ) writes text to the file
e.g.
```

```
f = open("myfile.txt", "a")
f.write("Hello World!")
f.close()
f = open("myfile.txt", "r")
print(f.read())
# Output:  Hello World!
```

---

open(“fileName”, “r”) syntax:

fileObj.open(“fileName”, “r”)
fileObj.read( )

```
# Open a file by using “r” mode for reading a file
# f.read( ) reads the contents of the file
e.g.
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
f = open("myfile.txt", "r")
print(f.read())
# Output:  Hello World!
```

---

open(“fileName”, “w”) syntax:

fileObj.open(“fileName”, “w”)
fileObj.write(“text”)

```
# Open a file by using “w” mode for writing a file
# f.write( “text” ) writes text to the file
e.g.
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
f = open("myfile.txt", "r")
print(f.read())
# Output:  Hello World!
```

---

or syntax:

operand1 or operand2

```
# Return true if one of the operands is true
```

e.g.

```
x = (6<10 or 8>9)
print(x)
# Output: True
```

---

ord() syntax:

ord('character')

Return a Unicode code of a specific character.

e.g.

```
print(ord('A'))
# Output: 65
```

---

os.chdir() syntax:

os.chdir(path)

Change a directory

e.g.

```
os.chdir("c:\\mydir")    # Set “mydir” as a current working directory
cwd = os.getcwd()
print("The current working directory is:", cwd)
# Output:  The current working directory is: c:\\mydir
```

---

os.getcwd syntax:

path = os.getcwd()

Return the current working directory

e.g.

```
os.chdir("c:\\mydir")    # Set “mydir” as a current working directory
cwd = os.getcwd()
print("The current working directory is:", cwd)
# Output:  The current working directory is: c:\\mydir
```

---

os.listdir syntax:

os.listdir(path)

Display all directories and files in the path

e.g.

```
import os
path = "/"
```

```
contents = os.listdir(path)
print("The directories and files in '", path, "' :")
print(contents)
# Output:  The directories and file in ' / ' :
['opt', 'run', 'tmp', 'lib64', 'dev', 'mnt', 'usr',……
```

---

overriding syntax:

```
class BaseClass
    def methodName():      # base method
……
class DerivedClass(BaseClass):
    def methodName():      # derived method
……
# When a method name in the derived class is the same as the method name
in base class, it is known as “overriding base method”
e.g.
class Computer:      # define a base class
    def __init__(self, name):   # define a constructor
        self.name = name
    def capacity(self, harddrive, memory):    # base method
        self.harddrive = harddrive
        self.memory = memory
class Laptop(Computer):     # define a derived class
    def capacity(self, harddrive, memory):    # derived method
        print (self.name)
        print ("Harddrive capacity: " + str(harddrive))
        print ("Memory capacity: "+ str(memory))
L = Laptop("Laptop")       # creates an object "L"
L.capacity( 8000, 6 )    # call the derived method capacity( )
# Output:
Laptop
Harddrive capacity: 8000
Memory capacity: 6
```

---

partition() syntax:

str.partition(text)

```
# Using the parameter 'text' to separate the string into three parts, return a tuple: (head, text, tail)
e.g.
str = "Visual Basic in 8 Hours"
s = str.partition("in")     # separate the string into three parts with "in"
print(s)
# Output:  ('Visual Basic ', 'in', ' 8 Hours')
```

---

pass syntax:

pass

```
# "pass" works as a placeholder; it does nothing currently.
e.g.
num = 100
if num > 100:
    pass
print('Here has a placeholder for future code')
# Output:   Here has a placeholder for future code
```

---

pattern syntax:

re.match(pattern, string)

```
# Match a pattern from the beginning of the string, return an object if true. Otherwise, return noon.
# span() returns the start index and ending index of the match.
# Note: match() only makes a match from the beginners
e.g.
import re
print(re.match('ray', 'ray@yahoo.com').span())
# Output:   (0, 3)
```

---

pattern.match(string) syntax:

pattern.match(string)

```
# Match the pattern with string, return true or false
e.g.
import re       # import re module
pattern = re.compile("^(\d{3})-(\d{3})-(\d{4})$")    # return a pattern
phoneNumber = input("Enter your phone number:")
```

```
valid = pattern.match(phoneNumber)      # match
print (phoneNumber)
if valid:
    print ("Valid Phone Number!")
# Output:
Enter your phone number:  123-123-1234
Valid Phone Number!
```

---

polymorphism syntax:

```
class MyClass1
    def myfunction(self):
    ……
class MyClass2
    def myfunction(self):
    ……
# Polymorphism means that the different objects executes different methods in different classes, but all methods use the same name.
e.g.
class Dog:              # define a class
    def cry(self):      # define a cry() method
        print ("Dog cries: Wou! Wou!")
class Cat:              # define a class
    def cry(self):      # define a cry() method
        print ("Cat cries: Meo! Meo!")
d = Dog()     # create an object "d" of the Dog class
d.cry()
c = Cat()     # create an object "c" of the Cat class
c.cry()
# Output:
Dog cries: Wou! Wou!
Cat cries: Meo! Meo!
```

---

pop() syntax:

```
set.pop()
# Remove a random item from the set
```

e.g.
```
animals = {"ass", "bat", "cow"}
animals.pop()
print(animals)
# Output:  {'ass', 'cow'}
```
---

pop() syntax:

list.pop(index)

Remove the element of a list at a specified index

e.g.
```
animals = ["ass", "bat", "cow"]
animals.pop(1)
print(animals)
# Output:  ['ass', 'cow']
```
---

popleft() syntax:

dequeObject.popleft()

Remove an item from left end of the deque

e.g.
```
from collections import deque
dq = deque([1,2,3,4,5])
dq.popleft()
print(dq)
# Output:   deque([ 2, 3, 4, 5])
```
---

popitem() syntax:

dictionary.popitem()

Remove the last item of a dictionary

e.g.
```
student = {"name": "Andy", "age": "17", "id": "0026"}
print("The removed item is:", student.popitem())
# Output:  The removed item is:  ('id', '0026')
```
---

pow() syntax:

math.pow( );

Return the first argument raised to the power of the second argument.

e.g.

```
import math
print math.pow(4,2)
# Output:  16.0
```

---

print() format syntax:

print(" %s %d %c %f %h %o" % ('string', digit, ascii, float, hex, oct))

```
# 'string' value will replace %s, 'digit' value will replace %d, …and so on
# Return a formatted string
```

e.g.

```
print ("My name is %s and age is %d" % ('Ray', 38))
# Output:  My name is Ray and age is 38
```

---

print() syntax:

print( )

```
# Print content or string to the screen
```

e.g.

```
print("Hello World!")
# Output:  Hello World!
```

---

product() syntax:

product(list1, list2)

```
# Return a cartesian product of multiple iterables
```

e.g.

```
from itertools import product
list1 = [1, 2, 3]
list2 = [5, 6, 7]
print(list(product(list1, list2)))
# Output:  [(1, 5), (1, 6), (1, 7), (2, 5), (2, 6), (2, 7), (3, 5), (3, 6), (3, 7)]
```

---

property() syntax:

```
def getProperty(self):
    return self.property     # get property
def setProperty(self, value):
```

```
    self.Property = value     # set property
def delProperty(self):
    del self.Property     # delete property
property(getProperty, setProperty, delProperty)
# Get a property, Set a property, Delete a property
e.g.
class Student:
    def __init__(self, name):
        self._name = name
    def getName(self):
        print('Getting a name:')
        return self._name
    def setName(self, value):
        print('Setting a name: ' + value)
        self._name = value
    def delName(self):
        print('Deleting a name')
        del self._name
    name = property(getName, setName, delName )
s = Student('Rose')
print(s.name)
s.name = 'Nancy'
del s.name
# Output:
Getting a name:
Rose
Setting a name: Nancy
Deleting a name
```

---

raise syntax:

```
raise Exception('message')
# Raise an exception and stop running the program
e.g.
num = -10
if num < 0:
    raise Exception("Sorry, the number cannot be negative!")
```

```
# Output:  Error message……
            Sorry, the number cannot be negative!
```

---

range syntax:

for var in range(n)

```
# Generate a sequence from 0 to n-1.
e.g.
for var in range(6):
    print(var)
# Output:    0,1,2,3,4,5.
```

---

range syntax:

for var in range(n1, n2)

```
# Generates a sequence from n1 to n2-1.
e.g.
for num in range(3,10) :
    print(num)
Output:     3,4,5,6,7,8,9.
```

---

randrange() syntax:

randrange(num1, num2)

```
# Return a random integer from num1 to num2
e.g.
import random
print(random.randrange(0, 8))
# Output:   6
```

---

read() syntax:

fileObj = open(“fileName”, “r”)
fileObj.read( )

```
# Open a file by using “r” mode for reading a file
# read( ) reads the contents of the file
e.g.
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
```

```
f = open("myfile.txt", "r")
print(f.read())
# Output:  Hello World!
```

---

readable() syntax:

fileObj.readable()

```
# Return true if the file is readable
e.g.
f = open("myFile.txt", "r")
print(f.readable())
# Output:  True
```

---

readline() syntax:

fileObj.readline()

```
# Read one line from the file
e.g.
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
f = open("myfile.txt", "r")
print(f.readline())
# Output:  Hello World!
```

---

readlines() syntax:

fileObj.readlines()

```
# Read all lines of the file and return a list containing the lines
e.g.
f = open("myfile.txt", "w")
f.writelines("Hello World! \n Greeting from: \n Ray Yao")
f.close()
f = open("myfile.txt", "r")
print(f.readlines())
# Output:  ['Hello World! \n', ' Greeting from: \n', ' Ray Yao']
```

---

reload() syntax:

reload(module_name)

Reload a module
e.g.
```
import sys
import importlib
importlib.reload(sys)
# Reload sys module
```
---

remove() syntax:

os.remove("file")
Remove a file
e.g.
```
import os
os.remove("myfile.txt")
print("myfile has been removed successfully!")
# Output:  myfile has been removed successfully!
```
---

remove() syntax:

collection.remove(element)
Remove a specified element of a collection
e.g.
```
animals = ["ass", "bat", "cow"]
animals.remove("bat")
print(animals)
# Output:  ['ass', 'cow']
```
---

rename() syntax:

os.rename("file1", "file2")
Rename a file name file1 as file2
e.g.
```
import os
os.rename( "file1.txt", "file2.txt" )
print("The file1 has been renamed to file2!")
# Output:  The file1 has been renamed to file2!
```
---

repeat symbol syntax:

*****

Match 0 or more characters in regular expression.

e.g.

re*

Match 0 or more characters, such as red, read, reader……

---

repeat() syntax:

itertools.repeat(num/str, times)

Repeat a number or a string for times

e.g.

```
import itertools
print (list(itertools.repeat("Ok", 3)))    # print "Ok" 3 times
# Output:  ['Ok', 'Ok', 'Ok']
```

---

replace() syntax:

replace(old, new)

Replace old character with new characters

e.g.

```
str = "Python is great!"
s = str.replace("great","very good")
print(s)
# Output:  Python is very good
```

---

repr() syntax:

repr(object)

Return a printable representation

e.g.

```
myList = [1, 2, 3, 4, 5]
print(repr(myList))
# Output:  [1, 2, 3, 4, 5]
```

---

return syntax:

return

Specify a value to be returned to the caller.

e.g.

```
def multiply ( n, m ) :
    return n*m
print(multiply( 2,100 ))
# Output:  200
```

---

reverse() syntax:

list.reverse()

```
# Reverse the element's sequence of a list
e.g.
animals = ["ass", "bat", "cow"]
animals.reverse()
print(animals)
# Output:  ['cow', 'bat', 'ass']
```

---

reversed() syntax:

reversed(object)

```
# Reverse the sequence of an object element
e.g.
str = "ABC"
r = reversed(str)
print(list(r))
# Output:  ['C', 'B', 'A']
```

---

rfind() syntax:

rfind(c)

```
# Same as find(), but find a character from right to left
e.g.
s = "JavaScript"
print(s.rfind("a"))
# Output:  3
```

---

rindex() syntax:

rindex(c)

```
# Same as index(), but find a character from right to left
e.g.
s = "total"
```

```
print(s.rindex("t"))
# Output:  2
```

---

rjust() syntax:

rjust(w,f)

```
# Right adjust string with width w and fill with f
str = " This is a right-adjust example"
print(str.rjust(40, '$'))
# Output:  $$$$$$$$$$ This is a right-adjust example
```

---

rmdir() syntax:

os.rmdir("/path/directory")

```
# Remove "/path/directory"
e.g.
import os
os.rmdir( "mydir" )
print("mydir has been removed successfully!")
# Output:  mydir has been removed successfully!
```

---

rotate() syntax:

dequeObject.rotate(n)

```
#  Rotate items in the deque object for n times
# Append items from left end, and pop items from right end.
e.g.
from collections import deque
dq = deque([1,2,3,4,5])
dq.rotate(2)     # rotate 2 times
print(dq)
# Output:  deque([ 4, 5, 1, 2, 3 ])
```

---

round() syntax:

round(float, decimal)

```
# Round a floating number
e.g.
print(round(0.555,2))
# Output:  0.56
```

---

rpartition() syntax:

str.rpartition("text")

Search for the last occurrence of the parameter "text", and return a tuple with three parts**:** (head, text, tail)

e.g.

```
str = "JAVA in 8 Hours books in Amazon"
s = str.rpartition("in")
print(s)
# Output:  ('JAVA in 8 Hours books', 'in', ' Amazon')
```

---

rsplit() syntax:

rsplit()

Separate each word of a string with commas, return a list

e.g.

```
str = "R in 8 Hours"
s = str.rsplit(" ")
print(s)
# Output:  ['R', 'in', '8', 'Hours']
```

---

rstrip() syntax:

rstrip()

Remove trailing spaces

e.g.

```
str = "      This is a rstrip sample!     "
print(str.rstrip( ))
# Output:          This is a rstrip sample!
```

---

run() syntax:

thread.run()

Run the current thread

e.g.

```
import time
import threading
def t1(v):
    print('Thread 1 value is: ', v)
```

```
def t2(v):
    print('Thread 2 value is: ', v)
thread1 = threading.Thread(target=t1, args=(10,))
thread2 = threading.Thread(target=t2, args=(20,))
thread1.run()
thread2.run()
# Output:       Thread 1 value is:  10       Thread 2 value is:  20
```

---

search() syntax:

re.search(pattern, string, flags=0)

```
# Scan the entire string and return the first successful match
# span() returns the start index and ending index of the match.
e.g.
import re
print(re.search('yahoo', 'ray@yahoo.com').span())
# Output:   (4, 9)
```

---

seek() syntax:

seek(index)

```
# Set the pointer to the specified index
e.g.
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
f = open("myfile.txt", "r")
f.seek(6)      # set the pointer at index 6
print(f.readline())
# Output:   World!
```

---

seekable() syntax:

fileObj.seekable()

```
# Return true if the file is seekable. ("seekable" means that the reading
position of the file can be changed.)
e.g.
f = open("myFile.txt", "r")
print(f.seekable())
```

Output: True

---

sendmail() syntax:

SMTP.sendmail(from_addr, to_addrs, msg[, mail_options, rcpt_options])

Send email by using SMTP

e.g.

```
obj = smtplib.SMTP('localhost')
obj.sendmail(sender, receivers, message.as_string())
```

obj.sendmail() sends an email

message.as_string() returns a complete message as string.

---

separator.join() syntax:

separator.join()

Join the strings by separators

e.g.

```
separator = "-";
str = ("x", "y", "z");
print (separator.join( str ))
```

Output: x-y-z

---

set environment variable syntax:

set PYTHONPATH=path

Set an environment variable in Unix

e.g.

```
set PYTHONPATH=/usr/local/lib/python
```

Note: The format of Unix path is: /…/…/…

Set an environment variable in Windows

e.g.

```
set PYTHONPATH=c:\python30\lib;
```

Note: The format of Windows path is: \…\…\…

---

set syntax:

setName = {“val1”, “val2”, “val3”}

Define a set, which is unordered, unchangeable, and unique

```
animal = { "ass", "bat", "cow"}
```

```
print(animal)
# Output:  {'cow', 'ass', 'bat'}
```

---

set add() syntax:

set.add(element)

Add an element to a set. If the element is already exists, it will not add any element.

e.g.

```
mySet = {1, 3, 5, 7, 9}     # define a set
mySet.add(8)
print(mySet)
# Output:  {1, 3, 5, 7, 8, 9}
```

---

set clear()

set.clear()

Clear all elements from a set

e.g.

```
mySet = {1,2,3,4,5}
# clear all elements
mySet.clear()
print(mySet)
# Output:  set()
```

---

set copy() syntax:

set2 = set1.copy()

Copy the elements of set1 to set1

e.g.

```
set1 = {10, 20, 30}
set2 = set1.copy()
print("The set2 is: ", set2)
# Output:  The set2 is:  {10, 20, 30}
```

---

set difference() syntax:

set1.difference(set2)

Get difference between set1 and set2, return elements which are unique in the set1.

e.g.
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print(**set1.difference(set2)**)
# Output: {1, 2}

---

set difference_update() syntax:

set1.difference_update(set2)
Get difference between set1 and set2, return elements which are unique in the set1, and update set1.
e.g.
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
set1.difference_update(set2)
print(" set1 = ", set1)
# Output: set1 = {1, 2}

---

set discard() syntax:

set.discard(element)
Remove a specified element, and return the remaining set.
e.g.
mySet = {1, 2, 3, 4, 5}
mySet.discard(3)
print(mySet)
# Output: {1, 2, 4, 5}

---

set intersection() syntax:

set1.intersection(set2)
Return common elements between set1 and set2.
e.g.
set1 = {1, 3, 5, 7}
set2 = {5, 7, 9}
print(**set1.intersection(set2)**)
# Output: {5, 7}

---

set intersection_update() syntax:

set1.intersection(set2)

Return common elements between set1 and set2, and update set1.

e.g.

```
set1 = {1, 3, 5, 7}
set2 = {5, 7, 9}
set1.intersection_update(set2)
print(" Set1= ", set1)
# Output:  Set1=  {5, 7}
```

---

set isdisjoint() syntax:

set1.isdisjoint(set2)

Return true if there is no any common element betweenset1 and set2, return false if there is common element between them.

e.g.

```
set1 = {0, 1, 2}
set2 = {3, 4, 5}
print(set1.isdisjoint(set2))
# Output:  true
```

---

set issubset() syntax:

set1.issubset(set2)

Return true if set1 is the subset of set2, return false if not.

e.g.

```
set1 = {0, 1, 2}
set2 = {0, 1, 2, 3, 4, 5}
print(set1.issubset(set2))
# Output:  true
```

---

set issuperset() syntax:

set1.issuperset(set2)

Return true if set1 is a super set of set2

e.g.

```
set1 = {"ant", "bee", "cat", "dog", "ewe"}
set2 = {"ant", "bee", "cat"}
ss = set1.issuperset(set2)
print(ss)
```

```
# Output: True
```

---

set pop() syntax:

```
item = Set.pop()
# Randomly remove an item form a set, and return a removed item.
e.g.
mySet = {'a', 'b', 'c', 'd', 'e'}
deleted_item = mySet.pop()
print("The removed item is: ", deleted_item)
# Output: The removed item is: d
```

---

set remove() syntax:

```
set.remove('element')
# Remove a specified element from a set
e.g.
fruit = {'apple', 'banana', 'cherry'}
fruit.remove('banana')     # remove banana
print(fruit)
# Output: {'apple', 'cherry'}
```

---

set symmetric_difference() syntax:

```
set1.symmetric_difference(set2)
# Return the different items between set1 and set2
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cat"}
diff = set1.symmetric_difference(set2)
print(diff)
# Output: {'bat', 'ant', 'ass', 'bee'}
```

---

set symmetric_difference_update() syntax:

```
set1.symmetric_difference_update(set2)
# Return the different items between set1 and set2, and update set1.
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cat"}
```

```
set1.symmetric_difference_update(set2)
print("set1 = ", set1)
# Output:
set1 = {'ass', 'bee', 'bat', 'ant'}
```

---

set union() syntax:

set1.union(set2)

Return all elements from two or multiple sets, but the elements are not duplicated in the result.

e.g.

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print('set1 U set2 = ', set1.union(set2))
# Output:  set1 U set2 =  {1, 2, 3, 4, 5, 6}
```

---

set update() syntax:

set1.update(Set2, Set3,…)

Update Set1 with Se2, Set3,…

e.g.

```
set1 = {2, 4, 6}
set2 = {1, 3, 5}
set3 = {7, 8}
print('Old set1:', set1)
set1.update(set2, set3)
print('New set1:', set1)
# Output:
Old set1: {2, 4, 6}
New set1: {1, 2, 3, 4, 5, 6, 7, 8}
```

---

set() syntax:

set(collection)

Return an unsorted set

e.g.

```
s = set(["ant", "bee", "cat"])
print(s)
# Output:  {'cat', 'ant', 'bee'}
```

---

setattr() syntax:

setattr(Class, object, attribute)

Set an attribute of an object in a class

e.g.

```
class Student:
    name = "Andy"
setattr(Student, 'score', 100)    # set attribute
a = getattr(Student, 'score')    # get attribute
print(a)
# Output:  100
```

---

set-cookie syntax:

set-cookie: name=name; expires=date; path=path; domain=domain

"set-cookie" sets up the cookie in Python.

HTTP cookies are sent via http headers.

e.g.

Set-Cookie : name="myCookie";expires=Wed, 28 Aug 2020 18 : 30 : 00 GMT

Set up a Cookie

---

setdefault() syntax:

dict.setdefault(key, default_value)

Return the item's value if the key is in the dictionary.

Return the default value if the key is not in the dictionary.

e.g.

```
dict = {'name': 'Andy', 'id': '0026'}
print ("id =", dict.setdefault('id', None))
print ("age =", dict.setdefault('age', None))
# Output:    id = 0026     age = None
```

---

single quotes syntax:

str = ‘string’

Define a string by using a pair of single quotes

e.g.

```
str = 'I am a string enclosed by a pair of single quotes'
print(str)
# Output:  I am a string enclosed by a pair of single quotes
```

---

sleep() syntax:

time.sleep(seconds)

```
# Suspend running the code for the given number of seconds.
e.g.
import time
import threading
def t1():
    time.sleep(3)      # thread1 sleep 3 seconds
    print('Thread1 prints later')
def t2():
    print('Thread2 prints first')
thread1 = threading.Thread(target=t1)
thread2 = threading.Thread(target=t2)
thread1.start()
thread2.start()
# Output:    Thread2 prints first    Thread1 prints later
```

---

slice() syntax:

slice(start, end, step)

```
# Return a sliced object
e.g.
list = ("a", "b", "c", "d", "e", "f", "g", "h")
items = slice(0, 7, 2)
print(list[items])
# Output:  ('a', 'c', 'e', 'g')
```

---

slicing syntax:

str[index1: index2]

```
# Return a range of characters from index1 to index2-1
# Using negative index starts slicing from the end of the string.
e.g.
```

```
myStr = "Shell Scripting in 8 Hours"
print(myStr[ 6 : 12 ])
# Output: Script
e.g.
myStr = "Shell Scripting in 8 Hours"
print(myStr[ -6 : -1 ])
# Output: Hour
```

---

smtp syntax:

```
smtpObj = smtplib.SMTP( [host])
# Create smtp object
e.g.
import smtplib
obj = smtplib.SMTP('localhost')
obj.sendmail(sender, receivers, message.as_string())
# obj.sendmail() sends email
```

---

smtp.sendmail() syntax:

```
SMTP.sendmail(from_addr, to_addrs, msg[, mail_options,
rcpt_options])
# Send email by using SMTP
e.g.
import smtplib
obj = smtplib.SMTP('localhost')
obj.sendmail(sender, receivers, message.as_string())
# obj.sendmail() sends email
```

---

socket() syntax:

```
import socket
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Create a socket object
# AF_INET means the address-family ipv4.
# SOCK_STREAM means connection-oriented TCP protocol.
e.g.
import socket # for socket
import sys
```

```
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print ("Socket is created successfully! ")
# Output:  Socket is created successfully!
```

---

sort() syntax:

list.sort()

```
# Sort all elements of a list
e.g.
animals = ["ass", "cow", "bat", "cat", "ant", "bee"]
animals.sort()
print(animals)
# Output:  ['ant', 'ass', 'bat', 'bee', 'cat', 'cow']
```

---

sorted() syntax:

sorted(list)

```
# Return a sorted list
e.g.
list = ("f", "e", "h", "c", "a", "g", "d", "b")
sl = sorted(list)
print(sl)
# Output:  ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

---

split() syntax:

split(separator)

```
# Split a string at a specified separator (at whitespace by default)
e.g.
str = "Python is a very good language"
print(str.split())     # split at every string whitespaces
 # Output:  ['Python', 'is', 'a', 'very', 'good', 'language']
```

---

splitlines() syntax:

splitlines()

```
# Separate a string with commas based on the "\n", return a list
e.g.
str = "Html Css\n in 8 Hours\n Book"
s = str.splitlines()
```

```
print(s)
# Output:  ['Html Css', ' in 8 Hours', ' Book']
```

---

sqrt() syntax:

math.sqrt( );

```
# Return the square root of the argument.
e.g.
import math
print(math.sqrt(4))
# Output:   2.0
```

---

start() syntax:

thread.start()

```
# Start a new thread
e.g.
import time
import threading
def t1(v):
    print('Thread 1 value is: ', v)
def t2(v):
    print('Thread 2 value is: ', v)
thread1 = threading.Thread(target=t1, args=(10,))
thread2 = threading.Thread(target=t2, args=(20,))
thread1.start()
thread2.start()
# Output:       Thread 1 value is:  10       Thread 2 value is:  20
```

---

startswith() syntax:

startswith()

```
# Returns true if the string starts with the specified character
e.g.
str = "Pandas in 8 Hours"
s = str.startswith("Pandas")
print(s)
# Output:  True
```

---

staticmethod() syntax:

staticmethod(Class.method)

```
# Converts a regular method to a static method
# Static methods are referenced by a class rather than its object.
e.g.
class Calculate:
    def add(n1, n2):
        return n1 + n2
Calculate.add = staticmethod(Calculate.add)
sum = Calculate.add(100, 68)
print('Sum:', sum)
# Output:  Sum: 168
```

---

str() syntax:

str(number)

```
# Covert a number to a string
e.g.
print(str(128))
# Output:  128
```

---

strftime() syntax:

time.strftime(format[, t])

print time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())

```
# Return a formatted date
# Please reference "Strftime Format Chart" in Appendix.
e.g.
from datetime import datetime
obj = datetime.now()
year = obj.strftime("%Y")
print("year:", year)
# Output:    year: 2018
```

---

str.swapcase() syntax:

str.swapcase()

```
# Swap the letter case of the string
```

e.g.
```
str = "jQUERY"
print(str.swapcase())
# Output:  Jquery
```
---

str.zfill(length) syntax:

str.zfill(length)

Add zeros to the left of the string with length

e.g.
```
str = "Python"
print(str.zfill(10))
# 0000Python
```
---

string[index] syntax:

string[index]

Return a character of the string at the specified index

e.g.
```
str = "Python"
print(str[2])
# Output:  t
```
---

string[index1: index2] syntax:

string[index1: index2]

Return characters from index1 to index2-1

e.g.
```
myString = "Python is a good language"
print(myString[7:16])
# Output:  is a good
```
---

strip() syntax:

strip()

Remove leading and trailing spaces

e.g.
```
str = "      This is a strip sample!     ";
print(str.strip( ))
# Output:    This is a strip sample!
```

---

sub() syntax:

re.sub(pattern, replace, string)

Substitute a match in a string

e.g.

```
import re
tel = "123-456-7890"
num = re.sub(r'\D', "", tel)       # remove "-"
print ("The phone is : ", num)
# Output:  The phone is : 1234567890
```

---

substring syntax:

string[index1: index2]

Subtract a substring from index1 to index2

e.g.

```
myString = "Python is a good language"
print(myString[7:16])
# Output:  is a good
```

---

sum() syntax:

sum(collection)

sum all items of a collection

e.g.

```
list = (2, 3, 4, 5, 6)
print(sum(list))
# Output:  20
```

---

super() syntax:

super().parent_method

An object that can access the method of the parent class

e.g.

```
class Parent:
    def __init__(self, word):
        self.greeting = word
    def say(self):
        print(self.greeting)
```

```
class Child(Parent):
    def __init__(self, word):
        super().__init__(word)
obj = Child("How are you doing!")
obj.say()
# Output:  How are you doing!
```

---

swapcase() syntax:

str.swapcase()

```
# Swap the letter case of the string
e.g.
str = "jQUERY"
print(str.swapcase())
# Output:   Jquery
```

---

symmetric_difference() syntax:

set1.symmetric_difference(set2)

```
# Return the different items between set1 and set2
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cat"}
diff = set1.symmetric_difference(set2)
print(diff)
# Output:  {'bat', 'ant', 'ass', 'bee'}
```

---

symmetric_difference_update() syntax:

set1.symmetric_difference_update(set2)

```
# Return the different items between set1 and set2, and update set1.
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cat"}
set1.symmetric_difference_update(set2)
print("set1 = ", set1)
# Output:
set1 = {'ass', 'bee', 'bat', 'ant'}
```

---

tee() syntax:

iterators = tee(iterable, number)

Return multiple independent iterators from a single iterable.

e.g.

```
from itertools import tee
myList = [1, 2,  3 , 4]
iterators = tee(myList, 3)     # create 3 iterators
for a, b, c, d in iterators:
    print(a, b, c, d)
```

Output:

```
1 2 3 4
1 2 3 4
1 2 3 4
```

---

tell() syntax:

fileObj.tell()

Return the current position of the file read/write pointer

e.g.

```
f = open("myFile.txt", "r")
print(f.tell())
```

Output: 28

---

thread creating syntax:

thr = threading.Thread(target=…, name="…", args=(...,))

Create a new thread

"target=myFunction" calls myFunction

name="myThread" defines the thread name as "myThread"

args=(parameter,) passes the parameter to myFunction

e.g.

```
import time
import threading
def func(n):
    print("Thread", n, " is running...")
thr = threading.Thread(target=func, name="Thread1", args=(1,))
thr.start()
print("The new thread name is: ", thr.name)
```

```
# Output:  Thread 1  is running...
                    The new thread name is:  Thread1
```

---

thread daemon syntax:

thread.daemon
thread.isDaemon

```
# Return True if the current thread is daemon thread
# "daemon thread" is a background thread, which is helpful for
implementing tasks that are not vital.
e.g.
from threading import *
def fun():
    print("The thread is running...")
thr = Thread(target=fun, daemon=True)    # create a daemon thread
print(thr.daemon)
thr.start()
print(thr.isDaemon())
# Output:       True       The thread is running...       True
```

---

thread name syntax:

thread.name

```
# Return the name of the current thread
e.g.
import time
import threading
thr = threading.Thread(name="myThread")     # create a thread
thr.start()
print("The current thread name is: ", thr.name)
# Output:  The current thread name is:  myThread
```

---

time() syntax:

time.time()

```
# Return the seconds since Epoch
e.g.
import time
s = time.time()
```

```
print("The seconds since Epoch is: ", s)
# Output:  The seconds since Epoch is:  1680100739.6688068
```

---

timedelta() syntax:

datetime.timedelta(variable=number)

```
# Return a date, which is used to calculate the duration.
# "variable=number" means days=num, seconds=num, weeks=num,
microseconds=num, milliseconds=num, minutes=num, hours=num
e.g.
import datetime
today = datetime.date.today()
yesterday = today - datetime.timedelta(days=1)
print("Today:", today)
print("Yesterday:", yesterday)
# Output:
Today: 2023-03-29
Yesterday: 2023-03-28
```

---

timestamp() syntax:

nowObj .timestamp()

```
# Return a timestamp, the date and time of occurrence of an event.
e.g.
from datetime import datetime
now = datetime.now()      # create a now object
print(now.timestamp())
# Output:  1680112942.655897
```

---

timezone() syntax:

datetime.now(pytz.timezone('region'))

```
# Return a datetime based on the specified timezone.
# "region" format:  'US/Pacific', 'US/Central', 'Asia/Tokyo',
'America/New_York', 'Europe/Athens', 'Africa/Maseru' ……
# Before using 'pytz', we may download and install pytz module.
e.g.
from datetime import datetime
import pytz
```

```
pacific = datetime.now(pytz.timezone('US/Pacific'))
print('US pacific datetime: ', pacific)
# Output:  US pacific datetime:  2023-03-27 04:59:34.015524-07:00
```

---

today() syntax:

```
date.today()
# Return the date of today
e.g.
from datetime import date
today = date.today()
print("Today is:", today)
# Output:  Today is: 2023-03-29
```

---

today() syntax:

```
dt = datetime.datetime.today()    # create a datetime object
# Return the current date and time
e.g.
import datetime
dt = datetime.datetime.today()     # create a datetime object
d = dt.date()     # get the date object from datetime object
print(dt)
print(d)
# Output:
2023-03-29 15:27:20.682288
2023-03-29
```

---

translate() syntax:

```
translate()
# Return a translated string
e.g.
dict = {80:  83}    # replaces ascii 80(P) with ascii 83(S)
str = "Hi, Peter!"
print(str.translate(dict))
# Output:  Hi, Seter!
```

---

truncate() syntax:

fileObj.truncate(bytes)

Truncate the file content to the specified bytes.

e.g.

```
f = open("myfile.txt", "a")
f.truncate(10)     # truncate the file content to 10 bytes
f.close()
f = open("myfile.txt", "r")
print(f.read())
# Output:  Hello Worl
```

---

try - except syntax:

```
try:
……
except  XxxError as message:
……
```

“try block” contains the code that may cause an exception.

“except block” catches the error, and handles the exception.

e.g.

```
try:
    value = 100/0
except ValueError as message:
    print("Exception occurs!", message)
# Output:  Traceback (most recent call last):….value = 100/0
                ZeroDivisionError: division by zero
```

---

try - except - else syntax:

```
try:
……
except:
……
else:
……
```

“try block” contains the code that may cause an exception.

“except block” catches the error, and handles the exception.

"else block" will be executed if no error occurs.

e.g.

```
try:
    print("Good!")
except:
    print("An error occurs!")
else:
    print("No error occurs!")
# Output:    Good!    No error occurs!
```

---

tuple creating syntax:

tuple = (val1, val2, val3)

```
# Define a tuple
e.g.
tpl = ("Mon", "Tue", "Wed", "Thu")
print(tpl[1])
# output:  Tue
```

---

tuple count() syntax:

tuple.count('element')

```
# Count how many a specified elements in a tuple
e.g.
myTuple = ('a', 'b', 'c', 'd', 'd', 'd')
count = myTuple.count('d')    # count how many 'd'
print('The count of d is: ', count)
# Output:  The count of d is:  3
```

---

tuple index() syntax:

tuple.index('element')

```
# Get the index of an element in a tuple.
e.g.
myTuple = ('ant', 'bee', 'cat', 'dog')
index = myTuple.index('cat')
print("The index is: ", index)
# Output:  The index is:  2
```

---

tuple len() syntax:

len(tuple)
Get the length of a tuple. (Count how many items in a tuple)
e.g.
myTuple = ("apple", "banana", "cherry")
print(**len(myTuple)**)
# Output: 3
---

tuple() syntax:

tuple(collection)
Return a tuple
e.g.
t = tuple(("ant", "bee", "cat"))
print(t)
# Output: ('ant', 'bee', 'cat')
---

type() syntax:

type()
Return the data type of a variable
e.g.
s = "Shell Scripting in 8 Hours"
print(type(s))
# Output: <class 'str'>
---

union() syntax:

set1.union(set2)
Return a union set from two sets
e.g.
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cow"}
u = set1.union(set2)
print(u)
# Output: {'cow', 'bat', 'ant', 'ass', 'bee', 'cat'}
---

unlink() syntax:

os.unlink(path)
Remove a file.

e.g.

```
import os
path = "hisfile.txt"
os.unlink(path)
print("The file has been deleted successfully!")
# Output:  The file has been deleted successfully!
```

---

unpack collection syntax:

```
collection = ["element1", "element2", "element3"]
var1, var2, var3 = collection
# Extract the collection's values into variables.
```

e.g.

```
colors = ["red", "yellow", "green"]
x, y, z = colors
print(x)
print(y)
print(z)
# Output:    red    yellow    green
```

---

update() syntax:

```
set1.update(set2)
# Add all items from set2 to set1
```

e.g.

```
set1 = {"ant", "bee", "cat"}
set2 = {"ass", "bat", "cow"}
set1.update(set2)
print(set1)
# Output:  {'bee', 'ant', 'bat', 'ass', 'cat', 'cow'}
```

---

update() syntax:

```
dictionary.update(key, value)
# Update a dictionary by adding a key/value pairs
```

e.g.

```
Student = {
    "name": "Andy",
    "id": "0026",
```

```
}
Student.update({"score": "100"})
print(Student)
#  Output:     {'name': 'Andy', 'id': '0026', 'score': '100'}
```

---

upper() syntax:

upper()

```
# Convert a string to upper case
e.g.
str = "Hello, World!"
print(str.upper())
# Output:  HELLO, WORLD!
```

---

utcnow() syntax:

utc = datetime.utcnow() # Create an utc object

```
# Return the current utc date and time
e.g.
from datetime import datetime
utc = datetime.utcnow()     #create an utcnow object
print(utc.isoformat())     # return utc in iso format (YYYY-MM-DD)
# Output:   2023-03-29T17:49:06.653613
```

---

values() syntax:

dictionary.values()

```
# Return all values of a dictionary
e.g.
Student = {
    "name": "Andy",
    "id": "0026",
    "score":"100"
}
print(Student.values())
# Output:  dict_values(['Andy', '0026', '100'])
```

---

variable assignment syntax:

var1 = var2 =var3 = value

var1, var2, var3 = value1, value2, value3
Multi variable assignment
e.g.
```
a = b = c = 100
x, y, z = 'A', 'B', 'C'
print(b)
print(y)
# Output:    100    B
```
---

variable delete syntax:

del variable
Delete a variable
e.g.
```
del myVar
print("myVar is deleted successfully!")
# Output:  myVar is deleted successfully!
```
---

vars() syntax:

vars(object)
Return the attributes of an object
e.g.
```
class Example:
    def __init__(self, a = 10, b = 20):
        self.a = a
        self.b = b
obj = Example()
print(vars(obj))
# Output:  {'a': 10, 'b': 20}
```
---

webbrowser syntax:

webbrowser.open("url")
Open a specified web page
e.g.
```
import webbrowser
url = "http://www.amazon.com"
webbrowser.open(url)
```

Output: (Amazon web page)

---

weekday() syntax:

calendar.weekday(year, month, date)

Return the day of the week (0 is Monday)

e.g.

```
import calendar
print(calendar.weekday(2023, 3, 29))
```

Output: 2

---

while loop syntax:

while <test-expression> :
<statement>

“while loop” is used to execute blocks of code repeatedly.

e.g.

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello!")
```

Output: Hello! Hello! Hello!

---

with syntax:

with statement1 as fileObj
fileObj.statement2

Make handy to process a file or exception. Ensure closing resources.

e.g.

```
with open('myfile.txt', 'w') as f:
    f.write('Pandas in 8 Hours!')
print("The text has been written to the file successfully!")
```

Output: The text has been written to the file successfully!

---

write() syntax:

fileObj = open(“fileName”, “w”)
fileObj.write(“text”)

Open a file by using “w” mode for writing a file

f.write(“text”) writes text to the file

e.g.

```
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
f = open("myfile.txt", "r")
print(f.read())
# Output:  Hello World!
```

---

writable() syntax:

fileObj.writable()

```
# Return true if the file is writable
```

e.g.

```
f = open("myFile.txt", "w")
print(f.writable())
# Output:  True
```

---

writelines() syntax:

fileObj.writelines(list)

```
# Write a few lines of content to the file
```

e.g.

```
f = open("myfile.txt", "w")
f.writelines("Hello World! \n Greeting from: \n Ray Yao")
f.close()
f = open("myfile.txt", "r")
print(f.readlines())
# Output:  ['Hello World! \n', ' Greeting from: \n', ' Ray Yao']
```

---

yield syntax:

yield variable

```
# Suspend the current function's execution and return a yielded value back to the caller.
```

e.g.

```
def myGenerator():
    num = 1
    print('First print')
```

```
    yield num
    num += 1
    print('Second print')
    yield num
    num += 1
    print('Third print')
    yield num
for data in myGenerator():
    print(data)
# Output:
First print
1
Second print
2
Third print
3
```

---

zfill() syntax:

str.zfill(length)

```
# Add zeros to the left of the string with length
e.g.
str = "Python"
print(str.zfill(9))
# Output:  000Python
```

---

zip() syntax:

zip(collection1, collection2, …)

```
# Returns one tuple composed of some collections
e.g.
a = ("ant", "bee", "cat")
b = ("ass", "bat", "cow")
z=(zip(a, b))
print(tuple(z))
# Output:  (('ant', 'ass'), ('bee', 'bat'), ('cat', 'cow'))
```

Appendix

Python Keywords Chart

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Data Types Chart

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either True or False
Set	set, frozenset	hold collection of unique items

Arithmetic Operator Chart

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
//	Integer Division
**	Exponentiation

Assignment Operators Chart

Operators	Examples:	Equivalent:
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y
//=	x//=y	x=x//y
**=	x**=y	x=x**y

Comparison Operators Chart

Operators	Running
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal
! =	not equal

Logical Operators Chart

Operators	Equivalent
and	logical AND
or	logical OR
not	logical NOT

Logical Results Chart

True **and** True returns true	True **and** False returns False	False **and** False returns False
True **or** True returns True	True **or** False returns True	False **or** False returns False
not False returns True	**not** True returns False	

Bitwise Operators

Operator	Name	Description
&	AND	return 1 if two bits are 1
|	OR	return 1 if one of two bits is 1
^	XOR	return 1 if both bits are different
~	NOT	Inverts 1 to 0 or 0 to1
<<	left shift	shifted to left by number of bits
>>	right shift	shifted to right by number of bits

Convert Data Type Chart

Function	Operation
int(x)	convert x to an integer number
str(x)	convert x to a string
chr(x)	convert x to a character
float(x)	convert x to a floating point number
hex(x)	convert x to a hexadecimal string
oct(x)	convert x to a an octal string
round(x)	round a floating-point number x **.**
type(x)	detect x data type

Escape Characters Chart

Characters	Description
\\	escape backslash
\’	escape single quote
\”	escape double quote
\n	new line
\r	return
\t	tab

Operators Precedence Chart

Precedence from highest to lowest

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'and'
^ |	Bitwise exclusive `or' and regular `or'
<= < > >=	Comparison operators
<> == ! =	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Format String Chart

Specifier	Description
d	digital integer
f	float
s	string
o	octal value
x	hexadecimal value

e	exponential
%	“%formatted value” from %original value

List Functions Chart

Function	Operation
list . append(n)	Append n to the end of list
list . clear()	Removes all the elements from the list
list . copy()	copy one list to another list
list . count(n)	Count how many n
list extend(lst)	Append each item of lst to list
list . index(n)	Return the index of n
list . insert(i,n)	Insert n before index i
list . pop(i)	Remove & return the item at index i
list . remove(n)	Remove the n
list . reverse()	Reverse the sequence of list
list . sort()	Sort the element of list increasingly

Dictionary Functions Chart

Function	Operation
d . clear()	Remove all items of d
d . copy()	Copy all items of d
d . fromkeys()	Return a dictionary with specified keys and value
d . get(key)	Return the values with specified key
d . items( )	Return key-value pairs of d
d . keys()	Return keys of d
d . pop(key)	Remove key and return its value
d . popitem()	Remove the last inserted key-value pair
d . setdefault(k,v)	Set key-value to d
d . values()	Return values of d
d1 . update(d2)	Add key-value of d1 to d2

Tuple Functions Chart

Function	Operation
item in tpl	Return true if item is in the tuple
len(tpl)	Return length of the tuple
tpl . count(item)	Count how many item in tuple
tpl . index(item)	Return the index of item

Set Functions Chart

Function	Operation
set . add(n)	Add x to the set
set . clear()	Remove all the elements from the set
set . copy( )	Copy the set
set1 . difference(set2)	Return items in set1 not in set2
set1 . difference_update(set2)	Same as above function, and update set1
set . discard()	Remove the specified item
set1 . intersection(set2)	Return items in both sets
set1 . intersection_update(set2)	Same as above function, and update set1
set1 . isdisjoint(set2)	Return true if two sets have no intersection
set1 . issubset(set2)	Return true if set1 is a subset of set2
set1 . issuperset(set2)	Return true if set1 is a superset of set2
set . pop( )	Remove one random item
set . remove(n)	Remove the item n
set1 . symmetric_difference(set2)	Return different items between set1 & set2
set1 . symmetric_difference_update(set 2)	Same as above function, and update set1

set **.** union()	Return a set containing the union of sets
set **.** update(a, b, c)	Add a, b, c to the set

Collection Difference Chart

Structures	Descriptions
List	store multiple changeable values
Tuple	store multiple unchangeable values
Set	store multiple unique values
Dictionary	store multiple key : value pairs

Regular Expressions Chart

Operators	Matches
^	Matches beginning of the line .
$	Matches end of line .
.	Matches any single character .
[...]	Matches any single character in brackets .
[^ ...]	Matches any single character not in brackets
?	Matches 0 or 1 occurrence
+	Matches 1 or more occurrence
*	Matches 0 or more occurrences
{ n}	Matches exactly n number of occurrences
{ n, m}	Matches at least n and at most m occurrences
a | b	Matches either a or b .
(re)	Groups regular expressions

Operation Strings Chart

Operator	Description
+	concatenate strings together
*	repeat a string
[key]	return a character of the string
[key1 : key2]	return characters from key1 to key2-1
in	check a character existing in a string
not in	check a character not existing in a string
''' '''	describe a function, class, method…

Strftime Format Chart

Code	Example	Description
%a	Mon	Weekday as locale’s abbreviated name .
%A	Monday	Weekday as locale’s full name .
%w	6	Weekday as a decimal number, 0 is Sun, 6 is Sat
%d	09	Day of the month as a zero-padded decimal number .
%-d	9	Day of the month as a decimal number .
%b	Oct	Month as locale’s abbreviated name .
%B	March	Month as locale’s full name .
%m	08	Month as a zero-padded decimal number .
%-m	8	Month as a decimal number . (Platform specific)
%y	09	Year without century as a zero-padded decimal number .
%Y	2023	Year with century as a decimal number .
%H	08	Hour (24-hour clock) as a zero-padded decimal number .
%-H	8	Hour (24-hour clock) as a decimal number .
%I	08	Hour (12-hour clock) as a zero-padded decimal number .
%-I	8	Hour (12-hour clock) as a decimal number .
%p	AM	Locale’s equivalent of either AM or PM .
%M	08	Minute as a zero-padded decimal number .
%-M	8	Minute as a decimal number .
%S	08	Second as a zero-padded decimal number .
%-S	6	Second as a decimal number .
%f	000000	Microsecond as a decimal number, zero-padded on left .

%z	+0000	UTC offset in the form ±HHMM[SS[. ffffff]]
%Z	UTC	Time zone name (empty string if the object is naive) .
%j	068	Day of the year as a zero-padded decimal number .
%-j	68	Day of the year as a decimal number .
%U	36	Week number of the year (Sunday as the first day)
%W	35	Week number of the year (Monday as the first day)
%c	datetime	Datetime like this : Mon Oct 8 07 : 06 : 08 2023
%x	08/09/23	Locale’s appropriate date representation .
%X	08 : 09 : 06	Locale’s appropriate time representation .
%%	%	A literal '%' character .

Exceptions Chart

Exception	Description
ArithmeticError	Trigged when an error occurs in numeric calculations
AssertionError	Trigged when an assert statement fails
AttributeError	Trigged when attribute reference or assignment fails
EOFError	Trigged when the input() hits an "end of file" condition
FloatingPointError	Trigged when a floating point calculation fails
GeneratorExit	Trigged when a generator is closed
ImportError	Trigged when an imported module does not exist
IndentationError	Trigged when indentation is not correct
IndexError	Trigged when an index of a sequence does not exist
KeyError	Trigged when a key does not exist in a dictionary
KeyboardInterrupt	Trigged when the user presses Delete, Ctrl+c, Ctrl+z
LookupError	Trigged when an index or a key is not found .
MemoryError	Trigged when a program runs out of memory
NameError	Trigged when a variable does not exist
NotImplementedError	Trigged when requiring an inherited class to override
OSError	Trigged when a system related operation causes error
OverflowError	Trigged when the calculation result is too large
ReferenceError	Trigged when a weak reference object does

	not exist
RuntimeError	Trigged when an error occurs of unspecific exceptions
StopIteration	Trigged when next() of an iterator has no next value
SyntaxError	Trigged when a syntax error occurs
TabError	Trigged when indentation consists of tabs or spaces
SystemError	Trigged when a system error occurs
SystemExit	Trigged when the sys . exit() function is called
TypeError	Trigged when two different types are combined
UnboundLocalError	Trigged when an undefined variable is referenced
UnicodeError	Trigged when a unicode problem occurs
UnicodeEncodeError	Trigged when a unicode encoding problem occurs
UnicodeDecodeError	Trigged when a unicode decoding problem occurs
UnicodeTranslateError	Trigged when a unicode translation problem occurs
ValueError	Trigged when a specified data type has wrong value
ZeroDivisionError	Trigged when the second operator in a division is zero
AttributeError	Trigged when attribute reference or assignment fails

Is Functions Chart

Functions	Return
isalpha()	return true if all characters are letters
isdigit()	return true if all characters are digits
isdecimal()	return true if all characters are decimals
isalnum()	return true if all characters are numbers or letters
islower()	return true if all characters are lowercase
isupper()	return true if all characters are uppercase
istitle()	return true if the string is title-case string
isspace()	return true if the string contains only whitespace

Html Functions Chart

Functions	Description
feed()	Feed some text to the parser .
close()	Force processing of all buffered data
reset()	Reset the instance, lose all unprocessed data .
getpos()	Return current line number and offset .
get_starttag_text()	Return the text of the most recently opened start tag .
handle_starttag()	Handle the start tag of an element
handle_endtag()	Handle the end tag of an element
handle_startendtag()	Handle the start & end tag
handle_data()	Handle arbitrary data
handle_entityref()	Handle a named character reference
handle_charref()	Handle decimal and hexadecimal numeric character references
handle_comment()	Handle html comment
handle_decl()	Handle an html doctype declaration
handle_pi()	Handle a processing instruction
escape()	Convert the characters &, < and > to html-safe sequences
unescape()	Convert character references to unicode characters .

Built-in Functions Chart

Function	Description
abs()	Get the absolute value of a number
all()	Return True if all items in an iterable object are true
any()	Return True if any item in an iterable object is true
ascii()	Replace none-ascii characters with escape character
bin()	Get the binary version of a number
bool()	Get the boolean value of the specified object
bytearray()	Get an array of bytes
bytes()	Get a bytes object
callable()	Return True if the specified object is callable, otherwise False
chr()	Get a character from the specified Unicode code .
classmethod()	Converts a method into a class method
compile()	Get the specified source as an object, ready to be executed
complex()	Get a complex number
delattr()	Delete the specified attribute from an object
dict()	Get a dictionary
dir()	Get a list of the specified object's properties and methods
divmod()	Get the quotient and the remainder
enumerate()	Operate a collection and return it as an enumerate object
eval()	Evaluate and executes an expression
exec()	Execute the specified code (or object)
filter()	Use a filter function to exclude items in an iterable object
float()	Get a floating point number
format()	Format a specified value
frozenset()	Get a frozenset object
getattr()	Get the value of the specified attribute (property or

	method)
globals()	Get the current global symbol table as a dictionary
hasattr()	Return True if the specified object has the specified attribute
hash()	Get the hash value of a specified object
help()	Execute the built-in help system
hex()	Convert a number into a hexadecimal value
id()	Get the id of an object
input()	Get the user input
int()	Get an integer number
isinstance()	Return True if an object is an instance of another object
issubclass()	Return True if a class is a subclass of a specified object
iter()	Get an iterator object
len()	Get the length of an object
list()	Get a list
locals()	Get an updated dictionary of the current local symbol table
map()	Return an iterator with specified function applied to each item
max()	Get the largest item in an iterable
memoryview()	Get a memory view object
min()	Get the smallest item in an iterable
next()	Get the next item in an iterable
object()	Get a new object
oct()	Convert a number into an octal
open()	Open a file and returns a file object
ord()	Convert an integer representing the unicode of a character
pow()	Get the value of x to the power of y
print()	Print to the standard output device

property()	Get, set, or delete a property
range()	Get a sequence of numbers, starting from 0 and increase by 1
repr()	Get a readable version of an object
reversed()	Get a reversed iterator
round()	Get a rounded numbers
set()	Get a new set object
setattr()	Set an attribute (property/method) of an object
slice()	Get a slice object
sorted()	Get a sorted list
staticmethod()	Convert a method into a static method
str()	Get a string object
sum()	Get the sum of the items of an iterator
super()	Get an object that represents the parent class
tuple()	Get a tuple
type()	Get the type of an object
vars()	Get the __dict__ property of an object
zip()	Get an iterator from two or more iterators

String Methods Chart

Method	Description
capitalize()	Convert the first character to upper case
casefold()	Convert string into lower case
center()	Get a centered string
count()	Get the number of times a specified value occurs in a string
encode()	Get an encoded version of the string
endswith()	Return True if the string ends with the specified value
expandtabs()	Set the tab size of the string
find()	Search the string for a specified value and return the position
format()	Format specified values in a string
format_map()	Format specified values in a string
index()	Search the string for a specified value and return the position
isalnum()	Return True if all characters in the string are alphanumeric
isalpha()	Return True if all characters in the string are in the alphabet
isascii()	Return True if all characters in the string are ascii characters
isdecimal()	Return True if all characters in the string are decimals
isdigit()	Return True if all characters in the string are digits
isidentifier()	Return True if the string is an identifier
islower()	Return True if all characters in the string are lower case
isnumeric()	Return True if all characters in the string are numeric
isprintable()	Return True if all characters in the string are printable
isspace()	Return True if all characters in the string are whitespaces
istitle()	Return True if the string follows the rules of a title

isupper()	Return True if all characters in the string are upper case
join()	Convert the elements of an iterable into a string
ljust()	Get a left justified version of the string
lower()	Convert a string into lower case
lstrip()	Get a left trim version of the string
maketrans()	Get a translation table to be used in translations
partition()	Get a tuple where the string is parted into three parts
replace()	Get a string where a value is replaced with another value
rfind()	Search a string for a specified value and return the last position
rindex()	Search a string for a specified value and return the last position
rjust()	Get a right justified version of the string
rpartition()	Get a tuple where the string is parted into three parts
rsplit()	Split the string at the specified separator, and return a list
rstrip()	Get a right trim version of the string
split()	Split the string at the specified separator, and return a list
splitlines()	Split the string at line breaks and return a list
startswith()	Get true if the string starts with the specified value
strip()	Get a trimmed version of the string
swapcase()	Swap cases, lower case becomes upper case and vice versa
title()	Convert the first character of each word to upper case
translate()	Get a translated string
upper()	Convert a string into upper case
zfill()	Fill the string with a specified number of 0 values at the beginning

File Methods Chart

Method	Description
close()	Close the file
detach()	Get the separated raw stream from the buffer
fileno()	Get a number that represents the stream
flush()	Flush the internal buffer
isatty()	Check whether the file stream is interactive or not
read()	Get the file content
readable()	Check whether the file stream can be read or not
readline()	Get one line from the file
readlines()	Get a list of lines from the file
seek()	Seek the file position
seekable()	Check whether the file allows us to change the file position
tell()	Get the current file position
truncate()	Resize the file to a specified size
writable()	Check whether the file can be written to or not
write()	Write the specified string to the file
writelines()	Write a list of strings to the file

File Opening Mode Chart

The syntax to open a file looks like this **:**

```
open(filename, “mode”)
```

The modes are listed as follows **:**

modes	actions
r	open file for reading (default)
w	open file for writing
a	open file for appending
+	open file for reading & writing
b	open file in binary mode
t	open file in text mode

Math Methods Chart

Assume that the first parameter is x, the second parameter is y .

Method	Description
math . acos()	Get the arc cosine of a number
math . acosh()	Get the inverse hyperbolic cosine of a number
math . asin()	Get the arc sine of a number
math . asinh()	Get the inverse hyperbolic sine of a number
math . atan()	Get the arc tangent of a number in radians
math . atan2()	Get the arc tangent of y/x in radians
math . atanh()	Get the inverse hyperbolic tangent of a number
math . ceil()	Rounds a number up to the nearest integer
math . comb()	Get number of ways to choose k from n without repetition & order
math . copysign()	Return a float with the magnitude of x but the sign of y
math . cos()	Get the cosine of a number
math . cosh()	Get the hyperbolic cosine of a number
math . degrees()	Converts an angle from radians to degrees
math . dist()	Get the Euclidean distance between two points (p and q)
math . erf()	Get the error function of a number
math . erfc()	Get the complementary error function of a number
math . exp()	Get E raised to the power of x
math . expm1()	Get Ex - 1
math . fabs()	Get the absolute value of a number
math . factorial()	Get the factorial of a number
math . floor()	Rounds a number down to the nearest integer
math . fmod()	Get the remainder of x/y
math . frexp()	Get the mantissa and the exponent of a specified number
math . fsum()	Get the sum of all items in any iterable (tuple, array, list, etc .)

math . gamma()	Get the gamma function at a number
math . gcd()	Get the greatest common divisor of two integers
math . hypot()	Get the Euclidean norm
math . isclose()	Checks whether two values are close to each other, or not
math . isfinite()	Checks whether a number is finite or not
math . isinf()	Checks whether a number is infinite or not
math . isnan()	Checks whether a value is NaN (not a number) or not
math . isqrt()	Rounds a square root number downwards to the nearest integer
math . ldexp()	Get the inverse of frexp()
math . lgamma()	Get the log gamma value of a number
math . log()	Get a natural logarithm of number, a logarithm of number to base
math . log10()	Get the base-10 logarithm of a number
math . log1p()	Get the natural logarithm of 1+number
math . log2()	Get the base-2 logarithm of a number
math . perm()	Get number of ways to choose k from n with order & no repetition
math . pow()	Get the value of x to the power of y
math . prod()	Get the product of all the elements in an iterable
math . radians()	Converts a degree value into radians
math .remainder()	Return the remainder of x abouty .
math . sin()	Get the sine of a number
math . sinh()	Get the hyperbolic sine of a number
math . sqrt()	Get the square root of a number
math . tan()	Get the tangent of a number
math . tanh()	Get the hyperbolic tangent of a number
math . trunc()	Get the truncated integer parts of a number

Math Properties Chart

Property	Description
math . e	Get Euler's number (2 . 7182 ...)
math . inf	Get a floating-point positive infinity
math . nan	Get a floating-point NaN (Not a Number) value
math . pi	Get a pi number (3 . 1415 ...)
math . tau	Get the value of tau (6 . 2831 ...)

Date Functions Chart

Function	Description
ctime()	Get a string representing the date
date()	Return a date object of years, month, and days,
fromisocalendar()	Get a date according to the ISO calendar
fromisoformat()	Get a date object according to a date string
fromordinal()	Get a date object from the proleptic Gregorian ordinal
fromtimestamp()	Get a date object from the POSIX timestamp
isocalendar()	Get a tuple year, week, and weekday
isoformat()	Get the string representation of the date
isoweekday()	Get the day of the week as integer(Mon is 1 and Sun is 7)
today()	Get the current local date
toordinal()	Get the proleptic Gregorian ordinal of the date
weekday()	Get the day of the week as integer (Mon is 0 and Sun is 6)

Time Functions Chart

fromisoformat()	Get a time according to the time string
time()	Get a time object of hours, minutes, and seconds
timetuple()	Get an object of time **.** struct_time

Datetime Functions Chart

combine()	Get a datetime combining a specified date and time
datetime()	Get a datetime object of both date and time
now()	Get a datetime object of current local date and time **.**
strptime()	Get a datetime from a string according to a specified format
timedelta()	Create a timedelta object

timezone()	Get the time zone of a specified location
utcfromtimestamp()	Get a datetime according to the POSIX timestamp
utcnow()	Get a datetime with the current UTC date & time

Random Methods Chart

Method	Description
betavariate()	Get a random 0 ~ 1 float number according to Beta distribution
choice()	Get a random element from the given sequence
choices()	Get a list with a random selection from the given sequence
expovariate()	Get a random float number according to Exponential distribution
gammavariate()	Get a random float number according to Gamma distribution
gauss()	Get a random float number according to Gaussian distribution
getrandbits()	Get a number representing the random bits
getstate()	Get the current internal state of the random number generator
lognormvariate()	Get a random float number according to a log-normal distribution
normalvariate()	Get a random float number according to normal distribution
paretovariate()	Get a random float number according to Pareto distribution
randint()	Get a random number between the given range
random()	Get a random float number between 0 and 1
randrange()	Get a random number between the given range
sample()	Get a given sample of a sequence
seed()	Initialize the random number generator
setstate()	Restore the internal state of the random number generator
shuffle()	Take a sequence and returns the sequence in a random order
triangular()	Get a random float number between two given

	parameters
uniform()	Get a random float number between two given parameters
vonmisesvariate()	Get a random float number according to von Mises distribution
weibullvariate()	Get a random float number according to Weibull distribution

Statistics Methods Chart

Method	Description
harmonic_mean()	Evaluate the harmonic mean of the given data
mean()	Evaluate the mean (average) of the given data
mean()	Evaluate the mean (average) of the given data
median()	Evaluate the median (middle value) of the given data
median_grouped()	Evaluate the median of grouped continuous data
median_high()	Evaluate the high median of the given data
median_low()	Evaluate the low median of the given data
mode()	Evaluate the mode of the given numeric or nominal data
pstdev()	Evaluate the standard deviation from an entire population
pvariance()	Evaluate the variance of an entire population
stdev()	Evaluate the standard deviation from a sample of data
variance()	Evaluate the variance from a sample of data

Requests Methods Chart

Method	Description
delete()	Send a DELETE request to the specified url
get()	Send a GET request to the specified url
head()	Send a HEAD request to the specified url
patch()	Send a PATCH request to the specified url
post()	Send a POST request to the specified url
put()	Send a PUT request to the specified url
request()	Send a request of the specified method to the specified url

cMath Methods Chart

Method	Description
cmath . acos(n)	Get the arc cosine value of n
cmath . acosh(n)	Get the hyperbolic arc cosine of n
cmath . asin(n)	Get the arc sine of n
cmath . asinh(n)	Get the hyperbolic arc sine of n
cmath . atan(n)	Get the arc tangent value of n
cmath . atanh(n)	Get the hyperbolic arctangent value of n
cmath . cos(n)	Get the cosine of n
cmath . cosh(n)	Get the hyperbolic cosine of n
cmath . exp(n)	Get the value of E^n
cmath . isclose()	Check whether two values are close, or not
cmath . isfinite(n)	Check whether n is a finite number
cmath . isinf(n)	Check whether n is a positive or negative infinity
cmath . isnan(n)	Check whether n is NaN (not a number)
cmath . log(n[,base])	Get the logarithm of n to the base
cmath . log10(n)	Get the base-10 logarithm of n
cmath . phase()	Get the phase of a complex number
cmath . polar()	Convert a complex number to polar coordinates
cmath . rect()	Convert polar coordinates to rectangular form
cmath . sin(n)	Get the sine of n
cmath . sinh(n)	Get the hyperbolic sine of n
cmath . sqrt(n)	Get the square root of n
cmath . tan(n)	Get the tangent of n
cmath . tanh(n)	Get the hyperbolic tangent of n

cMath Properties Chart

Property	Description
cmath . e	Get Euler's number (2 . 7182 ...)
cmath . inf	Get a floating-point positive infinity value
cmath . infj	Get a complex infinity value

cmath . nan	Get floating-point NaN (Not a Number) value
cmath . nanj	Get a complex NaN (Not a Number) value
cmath . pi	Get a pi number (3 . 1415 ...)
cmath . tau	Get the value of tau (6 . 2831 ...)

Thread Property & Method Chart

Meth . Prop .	**Description**
daemon	A bool value indicating if the thread is a daemon thread
getName()	Get the name of a thread .
ident	A unique identifier for a running thread
isAlive()	Check whether a thread is still executing .
join()	Wait for joined threads to terminate .
lock	Prevent multiple threads from accessing same thing simultaneously
name	A string identifying the thread
native_id	The native integral thread id of the thread
run()	Word as the entry point for a thread .
setName()	Set the name of a thread .
start()	Start a thread by calling the run method .

Threading Methods Chart

Methods	**Description**
active_count()	Return the number of thread objects that are active .
current_thread()	Return the current thread object
enumerate()	Return a list of all threads that are currently active .
excepthook()	Process uncaught exceptions trigged by run()
get_ident()	Get the thread identifier of the current thread
get_native_id()	Get the native integral thread id of the current thread
main_thread()	Get the main thread object
setprofile()	Set a profile function for all threads from current module
settrace()	Set a trace function for all threads from current module
stack_size()	Get the thread stack size used while creating a new

	thread
thread()	Create a thread object of the Thread class

Ray Yao's Team
www.amazon.com/author/ray-yao

Disclaimer

This book is intended as a basic syntax manual only; it cannot include all entries of this subject**.** Its purpose is as a supplement for a cheat sheet book, not as a whole Python dictionary**.**

Paperback Books by Ray Yao

C# Cheat Sheet

C++ Cheat Sheet

JAVA Cheat Sheet

JAVASCRIPT Cheat Sheet

PHP MYSQL Cheat Sheet

PYTHON Cheat Sheet

HTML CSS Cheat Sheet

LINUX Command Line