Contents

[CoreyMS - List 4](#_Toc14039186)

[# append() 5](#_Toc14039187)

[# extend() - Add the elements of a list to the end of the current list 5](#_Toc14039188)

[# remove() – remove specified element from a list 5](#_Toc14039189)

[# pop() – removes last element of a list 5](#_Toc14039190)

[# reverse() 5](#_Toc14039191)

[# sort() 5](#_Toc14039192)

[# sorted() - sorting the list without altering the original list 6](#_Toc14039193)

[# w3schools 7](#_Toc14039194)

[CoreyMS - Tuple 8](#_Toc14039195)

[Socratica 9](#_Toc14039196)

[Test the size of list vs tuple 9](#_Toc14039197)

[Test how quick do tuples are made 9](#_Toc14039198)

[A list with 1 element 9](#_Toc14039199)

[An Alernative way of making tuples 10](#_Toc14039200)

[Compare following two examples 10](#_Toc14039201)

[CoreyMS - Sets 11](#_Toc14039202)

[Practical Examples 15](#_Toc14039203)

[Casting 16](#_Toc14039204)

[Creating empty lists, tuples & sets 16](#_Toc14039205)

[Dictionary 17](#_Toc14039206)

[# get(), Setting a default value, in our case 'Not Found', if key doesn't exists! 17](#_Toc14039207)

[# update(), can updated multiple key-value pairs 17](#_Toc14039208)

[# del 17](#_Toc14039209)

[# pop 17](#_Toc14039210)

[# keys() 17](#_Toc14039211)

[# values() 17](#_Toc14039212)

[# items() 17](#_Toc14039213)

[# pop() 18](#_Toc14039214)

[# popitem() - removes an arbitrary element (the same element which is returned) from the dictionary. 18](#_Toc14039215)

[# clear() 18](#_Toc14039216)

[Constructors 19](#_Toc14039217)

[Sort List, Tuple, Dictionary 20](#_Toc14039218)

[# SORTING LIST 20](#_Toc14039219)

[# sorted(), it returns an new sorted list 20](#_Toc14039220)

[# sort(), it returns nothing but sorts the original list 20](#_Toc14039221)

[# SORTING TUPLE 20](#_Toc14039222)

[# SORTING DICTIONARY 21](#_Toc14039223)

Help Chart – List of supported and unsupported methods of the following collections

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| List | Tuple | | Sets | Dictionary |
| sort() | **supported** | **unsupported** | intersection() | sorted() |
| sorted() | sorted() | sort() | intersection\_update() | keys() |
| len() | len() | append() | difference() | values() |
| append() | index() | extend() | difference\_update() | items() |
| insert() | count() | insert() | symmetric\_difference() | pop() |
| remove() | min(), max(), sum() | pop() | symmetric\_difference\_update() | popitem() |
| pop() | enumerate() | remove() | union() | clear() |
| extend() | join() | copy() | add() | sort() - not working |
| reverse() |  | reverse() | copy() |  |
| min(), max(), sum() |  | clear() | isdisjoint() |  |
| index() |  |  | issubset() |  |
| enumerate() |  |  | issuperset() |  |
| join() |  |  | clear() |  |
| copy() |  |  | discard() |  |
| count() |  |  | pop() |  |
| clear() |  |  | remove() |  |
|  |  |  | update() |  |

# CoreyMS - List

|  |  |  |
| --- | --- | --- |
| Data Stucture | Properties | |
| List | ordered, mutable | Allows duplicate |
| Tuple | ordered, immutable | Allows duplicate |
| Set | unordered | No duplicate |
| Dictionary | unordered | No duplicate |

|  |  |
| --- | --- |
| Methods | Examples |
| sort() | list.sort(reverse=true|false) |
| sorted() | returned\_val = sorted(list) |
| len() | len(list) |
| append() | list.append("elem\_name") |
| insert() | list.insert(index, "elem\_name") |
| remove() | list.remove("elem\_name") |
| pop() | list.pop() |
| extend() | list1.extend(list2) |
| reverse() | list.reverse() |
| min(), max(), sum() | min(list), max(list), sum(list) |
| index() | list.index("elem\_name") |
| enumerate() | enumerate(list, start=num) |
| join() | res = ", ".join(list) |
| copy() | list2 = list1.copy |
| count() | list.count("elem\_name") |
| clear() | list.clear() |
| list() | list(("apple", "banana", "elem\_name")) |

courses = ['Python', 'Django', 'Pandas', 'Numpy']

res1 = courses[0:2] # Index '0' included, index '2' excluded

# ['Python', 'Django']

res2 = courses[3:]

# ['Numpy']

## # append()

res3 = courses.append('Matplotlib')

# ['Python', 'Django', 'Pandas', 'Numpy', 'Matplotlib']

# insert() - adds an element at the user specified position

res4 = courses.insert(0, 'Deep Learning')

# ['Deep Learning', 'Python', 'Django', 'Pandas', 'Numpy', 'Matplotlib']

## # extend() - Add the elements of a list to the end of the current list

programming\_langs = ['JS', 'Java']

res5 = courses.extend(programming\_langs)

# ['Deep Learning', 'Python', 'Django', 'Pandas', 'Numpy', 'Matplotlib', 'JS', 'Java']

## # remove() – remove specified element from a list

res6 = courses.remove('Java')

# ['Deep Learning', 'Python', 'Django', 'Pandas', 'Numpy', 'Matplotlib', 'JS']

## # pop() – removes last element of a list

res7 = courses.pop()

# ['Deep Learning', 'Python', 'Django', 'Pandas', 'Numpy', 'Matplotlib']

## # reverse()

res8 = courses.reverse()

# ['Matplotlib', 'Numpy', 'Pandas', 'Django', 'Python', 'Deep Learning']

## # sort()

res9 = courses.sort(reverse=False) # ASC

# ['Deep Learning', 'Django', 'Matplotlib', 'Numpy', 'Pandas', 'Python']

res9\_1 = courses.sort(reverse=True) # DESC

# ['Python', 'Pandas', 'Numpy', 'Matplotlib', 'Django', 'Deep Learning']

## # sorted() - sorting the list without altering the original list

temp\_sorted = sorted(courses, reverse = True|False)

print(temp\_sorted)

# ['Deep Learning', 'Django', 'Matplotlib', 'Numpy', 'Pandas', 'Python']

# min(), max(), sum()

nums = [1, 2, 3, 4, 5, 6, 7]

print(min(nums)) # 1

print(max(nums)) # 7

print(sum(nums)) # 28

# index()

print(courses) # ['Python', 'Pandas', 'Numpy', 'Matplotlib', 'Django', 'Deep Learning']

print(courses.index('Django')) # 4

# in – membership testing

print('Python' in courses) #True

# for

for course in courses:

print(course)

# enumerate()

for index, course in enumerate(courses, start=1):

print(index, course)

# 1 Python

# 2 Pandas

# 3 Numpy

# 4 Matplotlib

# 5 Django

# 6 Deep Learning

# join()

courses\_str = '- '.join(courses)

# Python- Pandas- Numpy- Matplotlib- Django- Deep Learning

# split()

courses\_str = courses\_str.split('-')

# ['Python', ' Pandas', ' Numpy', ' Matplotlib', ' Django', ' Deep Learning']

## # w3schools

# copy()

thislist = ["apple", "banana", "cherry"]

mylist = thislist.copy()

print(mylist) # ['apple', 'banana', 'cherry']

# list constructor - list()

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets

print(thislist) # ['apple', 'banana', 'cherry']

# count() - counts numbers of occurences of an element in a list

fruits = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = fruits.count(9)

print(x) # 2

# concatenation

courses = ['Python', 'Django', 'Pandas', 'Numpy']

fruits = ["apple", "banana", "cherry"]

res = courses + fruits

print(res)

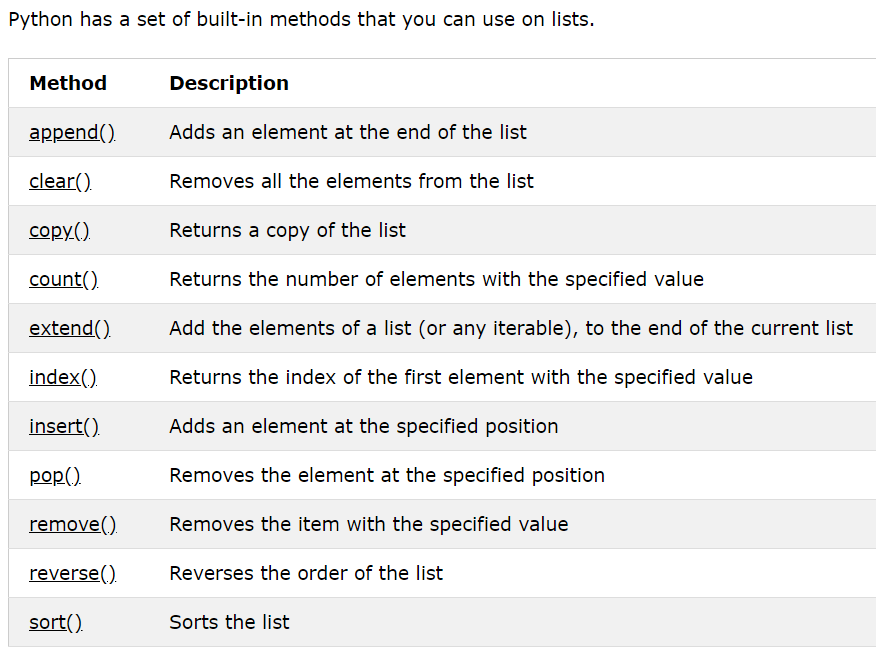
# ['Python', 'Django', 'Pandas', 'Numpy', 'apple', 'banana', 'cherry']

# Repetition

fruits = ["apple", "banana", "cherry"]

print(fruits\*2)

# ['apple', 'banana', 'cherry', 'apple', 'banana', 'cherry']



# CoreyMS - Tuple

*Tuples are mutable.*

Since tuples are immutable you can't use the following methods: append(), extend(), insert(), pop(), remove(), copy(), reverse(), sort(), clear()

The following methods have identical usage both in tuples and lists: index(), count(), len(), min(), max(), sum(), join(), enumerate(), sorted()

tuple\_1 = ('History', 'Math', 'Physics', 'CompSci')

print(tuple\_1) # ('History', 'Math', 'Physics', 'CompSci')

tuple\_1[0] = 'Art' # TypeError, since tuples are immutable

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# tuple constructor

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets

print(thistuple)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# concatenation

courses = ('Python', 'Django', 'Pandas')

fruits = ("apple", "banana", "cherry")

res = courses + fruits

print(res) # ('Python', 'Django', 'Pandas', 'apple', 'banana', 'cherry')

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Tuple | |
| supported methods | **unsupported methods** |
| sorted() | sort() |
| len() | append() |
| index() | extend() |
| count() | insert() |
| min(), max(), sum() | pop() |
| enumerate() | remove() |
| join() | copy() |
|  | reverse() |
|  | clear() |

|  |  |
| --- | --- |
| List | Tuple |
| Add data | Cannot be changed |
| Remove data | Immutable |
| Change data | Tuples are made quickly |

\_\_\_\_\_\_\_\_\_\_\_\_\_

courses = ('Python', 'Django', 'Pandas')

courses.append('JavaScript') # You cannot append anything to a tuple

# but there is a defacto alternative to append items in a tuple

courses = courses + ('JavaScript',)

print(courses) # ('Python', 'Django', 'Pandas', 'JavaScript')

\_\_\_\_\_\_\_\_\_\_\_\_\_

courses = 'Python', # this also creates a tuple, weird!

\_\_\_\_\_\_\_\_\_\_\_\_\_

# Socratica

## Test the size of list vs tuple

import sys

list1 = [1, 2, 'a', 'b', True, 234.23423]

tuple1 = (1, 2, 'a', 'b', True, 234.2342)

print("List size: ", sys.getsizeof(list1))

print("Tuple size: ", sys.getsizeof(tuple1))

# List size: 112

# Tuple size: 96

## Test how quick do tuples are made

import timeit # Measure execution time of small code snippets.

# lets measure execution time of the following list and tuple a million times

duration\_to\_make\_list = timeit.timeit(stmt="[1, 2, 3, 4, 5]", number=1000000)

duration\_to\_make\_tuple = timeit.timeit(stmt="(1, 2, 3, 4, 5)", number=1000000)

print(duration\_to\_make\_list)

print(duration\_to\_make\_tuple)

# 0.11285215400000001

# 0.013594196000000003

## A list with 1 element

res1 = ('Hello world')

print(res1) # this returns an string not enclosed within parenthesis

# Hello world

res2 = ('Hello world', ) # adding the extra comma make the output look like tuple

print(res2)

# ('Hello world',)

## An Alernative way of making tuples

test1 = 1,

test2 = 1, 2, 3

test3 = 1, 2, 3, 4, 5

print(type(test1), " => ",test1)

print(type(test2), " => ",test2)

print(type(test3), " => ",test3)

# <class 'tuple'> => (1,)

# <class 'tuple'> => (1, 2, 3)

# <class 'tuple'> => (1, 2, 3, 4, 5)

## Compare following two examples

**# First Example**

# (age, country, knows\_python)

survey1 = (19, 'Afghanistan', True)

age = survey1[0]

country = survey1[1]

knows\_python = survey1[2]

print('Age = ', age)

print('Country = ', country)

print('Knows Python = ', knows\_python)

# Age = 19

# Country = Afghanistan

# Knows Python = True

**# Second Example**

# (age, country, knows\_python)

age, country, knows\_python = (19, 'Afghanistan', True)

print('Age = ', age)

print('Country = ', country)

print('Knows Python = ', knows\_python)

# Age = 19

# Country = Afghanistan

# Knows Python = True

# CoreyMS - Sets

Sets are mutable and accepts no duplicate and they’re unordered. Use sets when order and frequency of data doesn’t matters.

We are refereing **x** as first set and **y** as the second set in Description section below:

|  |  |
| --- | --- |
| Sets | Description |
| intersection() | common items of both sets |
| intersection\_update() | update set with only common items |
| difference() | uncommon items of both sets |
| difference\_update() | update set with only uncommon items |
| symmetric\_difference() | all uncommon items of both sets |
| symmetric\_difference\_update() | update set with only uncommon items of both sets |
| union() | union both sets |
| add() | add an item to the set |
| copy() | copy set |
| isdisjoint() | returns true if no items in set x exists in set y |
| issubset() | is x sub-s et of set y |
| issuperset() | is x superset of set y |
| clear() | clear the set |
| discard() | removes item. it doesn’t throw any error if item doesn’t exist |
| pop() | removes last element from the list |
| remove() | remove item, it throws error if item exists |
| update() | insert items from one set to another |

# intersection(), common elements of both sets

x = {'History', 'Math', 'Physics', 'CompSci'}

y = {'art', 'Math', 'Physics', 'Java'}

z = {'Judo', 'Math', 'JavaScript'}

res1 = x.intersection(y, z)

res2 = x & y & z # identical to above

# {'Math'}

# intersection\_update() – Remove uncommun elements (the items that is not present in both x) and y sets

x = {'History', 'Math', 'Physics', 'CompSci'}

y = {'art', 'Math', 'Physics', 'Java'}

x.intersection\_update(y) # it updates the the list with intersected elements

print(x) # {'Physics', 'Math'}

# difference(), uncommon elements of both sets

cs\_courses = {'History', 'Math', 'Physics', 'CompSci'}

art\_courses = {'art', 'Math', 'Physics', 'Java'}

res2 = cs\_courses.difference(art\_courses)

res2 = cs\_courses – art\_courses # identical to above [edureka YT channel]

# {'History', 'CompSci'}

# difference\_update() - removes common elements of both sets

cs\_courses = {'History', 'Math', 'Physics', 'CompSci'}

art\_courses = {'art', 'Math', 'Physics', 'Java'}

cs\_courses.difference\_update(art\_courses)

print(cs\_courses) # {'History', 'CompSci'}

# symmetric\_difference() – returns all uncommon elements between two sets

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

z = x.symmetric\_difference(y) # or z = x ^ y

print(z) # {'google', 'microsoft', 'cherry', 'banana'}

# symmetric\_difference\_update() - remove common items AND insert uncommon items in set (in example below is "x")

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

x.symmetric\_difference\_update(y)

print(x) # {'microsoft', 'google', 'cherry', 'banana'}

**Note:** You can pass list as an argument in difference(), intersection() and … methods.

# union(), union both sets together

cs\_courses = {'History', 'Math', 'Physics', 'CompSci'}

art\_courses = {'art', 'Math', 'Physics', 'Java'}

res3 = cs\_courses|art\_courses # identical to above line

res3 = cs\_courses.union(art\_courses)

# {'CompSci', 'Java', 'History', 'Math', 'art', 'Physics'}

# add() - adds an element to the set

fruits = {"apple", "banana", "cherry"}

fruits.add("orange")

print(fruits) # {'apple', 'banana', 'cherry', 'orange'}

# clear() - removes all the elements from the set

this\_set = {"apple", "banana", "cherry"}

this\_set.clear()

print(this\_set) # set()

# copy()

fruits = {"apple", "banana", "cherry"}

x = fruits.copy()

print(x) # {"apple", "banana", "cherry"}

# isdisjoint() - returns True if no items in set x is present in set y

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "facebook"}

z = x.isdisjoint(y)

print(z) # True

# issubset() - returns True if all items of set x are present in set y

x = {"a", "b", "c"}

y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z) # True

# issuperset() - returns True if all items set y are present in set x

x = {"f", "e", "d", "c", "b", "a"}

y = {"a", "b", "c"}

z = x.issuperset(y)

print(z) # True

# pop() - removes last element from the list

fruits = {"apple", "banana", "cherry"}

fruits.pop()

print(fruits) # {"apple", "banana"}

# remove() – it throws an error if item doesn’t exist

fruits = {"apple", "banana", "cherry"}

fruits.remove("banana")

print(fruits)

# discard() – removes item. It doesn’t throw any error if item doesn’t exist

thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset) # {'apple', 'cherry'}

# update() - Insert the items from set y into set x:

* Update by set
* Update by list
* Update by set and list

# e.g001

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

x.update(y) # update by set

print(x) # {'apple', 'cherry', 'google', 'microsoft', 'banana'}

# e.g002

x = {"apple", "banana", "cherry"}

x.update(["google", "microsoft", "apple"]) # update by list

print(x) # {'apple', 'cherry', 'google', 'microsoft', 'banana'}

# e.g003

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

x.update(["corey", "traversy", "sentdex"], y) # update by list and set

print(x)

# {'traversy', 'banana', 'sentdex', 'google', 'apple', 'cherry', 'microsoft', 'corey'}

# Practical Examples

# PRACTICAL EXAMPLE (1): remove duplicate items from this list

list1 = [1, 2, 3, 4, 1, 3 , 4]

list2 = set(list1)

print(list2) # {1, 2, 3, 4} - but we want the output to be a list so:

list2 = list(set(list1))

print(list2) # [1, 2, 3, 4] - it returned a list and duplicated values are gone

# PRACTICAL EXAMPLE (2):

employees = ['Corey', 'Jim', 'Steven', 'April', 'Judy', 'Jenn', 'John', 'Jane']

gym\_members = ['April', 'John', 'Corey']

developers = ['Judy', 'Corey', 'Steven', 'Jane', 'April']

# e.g. Print list of Employees that are gym members as well

# Attention: we converted employees list to set then operate set methods on it

res1 = set(employees).intersection(gym\_members)

# {'April', 'John', 'Corey'}

# Ouput set of employees that are neither gym\_members nor developers

res2 = set(employees).difference(gym\_members, developers)

print(res1)

# {'Jenn', 'Jim'}

# Casting

members = ['Dave', 'John', 'Corey']

# Convert a list to a set

set\_mem = set(members)

print(set\_mem)

# {'John', 'Dave', 'Corey'}

# Convert a list to tuple

tuple\_mem = tuple(members)

print(tuple\_mem)

# ('Dave', 'John', 'Corey')

# Convert a tuple to list

tup = ('Dave', 'John', 'Corey')

list1 = list(tup)

print(list1)

# ['Dave', 'John', 'Corey']

# Convert a tuple to set

tup = ('Dave', 'John', 'Corey')

set1 = set(tup)

print(set1)

# {'Dave', 'John', 'Corey'}

# Creating empty lists, tuples & sets

# Empty Lists

empty\_list = []

empty\_list = list()

# Empty Tuples

empty\_tuple = ()

empty\_tuple = tuple()

# Empty Sets

empty\_set = {} # Don’t do like this, this isn't right! It creates a dict

empty\_set = set()

# Dictionary

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

## # get(), Setting a default value, in our case 'Not Found', if key doesn't exists!

print(student.get('phone', 'Not Found'))

# Not Found

## # update(), can updated multiple key-value pairs

res1 = student.update({'name':'Marry', 'age': 32, 'phone': '555-5999-6666'})

# {'name': 'Marry', 'age': 32, 'courses': ['Math', 'Chemistry'], 'phone': '555-5999-6666'}

|  |
| --- |
| Dictionary |
| sorted() |
| keys() |
| values() |
| items() |
| pop() |
| popitem() |
| clear() |

## # del

del student['courses']

# {'name': 'Marry', 'age': 32, 'phone': '555-5999-6666'}

## # pop

deleted\_key\_value = student.pop('age')

print(deleted\_key\_value) # 32

print(student)

# {'name': 'Marry', 'phone': '555-5999-6666'}

## # keys()

print(student.keys())

# dict\_keys(['name', 'phone'])

## # values()

print(student.values())

# dict\_values(['Marry', '555-5999-6666'])

## # items()

print(student.items())

# dict\_items([('name', 'Marry'), ('phone', '555-5999-6666')])

## # pop()

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

student.pop('name')

print(student)

# {'age': 23, 'courses': ['Math', 'Chemistry']}

## # popitem() - removes last key value pair of a dict

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

student.popitem()

print(student)

# {'name': 'John Doe', 'age': 23}

## # clear()

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

student.clear()

print(student)

# {}

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

for key in student:

print(key)

# name

# phone

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

for key in student:

key = key

value = str(student[key])

print(key + " : " + value)

# name : John Doe

# age : 23

# courses : ['Math', 'Chemistry']

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

for value in student.values():

print(value)

# Marry

# 555-5999-6666

student = {'name':'John Doe', 'age': 23, 'courses' : ['Math', 'Chemistry']}

for key, value in student.items():

print(key, ' - ', value)

# name - Marry

# phone - 555-5999-6666

# Constructors

thislist = list(("apple", "banana", "cherry"))

thistuple = tuple(("apple", "banana", "cherry"))

thisset = set(("apple", "banana", "cherry"))

thisdict =  dict(brand="Ford", model="Mustang", year=1964)

# keys are not enclosed within quotation marks, instead of colon there is equal

# Sort List, Tuple, Dictionary

## # SORTING LIST

nums = [9, 2, 4, 5, 1, 8, 6]

## # sorted(), it returns an new sorted list

s\_nums = sorted(nums, reverse=True) # it doesn't affect the original list

print(s\_nums) # [1, 2, 4, 5, 6, 8, 9]

print(nums) # [9, 2, 4, 5, 1, 8, 6]

## # sort(), it returns nothing but sorts the original list

nums.sort() # it affects on original list

print(nums) # [1, 2, 4, 5, 6, 8, 9]

# e.g. sort this list regardless of their signs

li = [-6, -5, -4, 1, 2, 3]

sorted\_li = sorted(li, key=abs)

print(sorted\_li) # [1, 2, 3, -4, -5, -6]

## # SORTING TUPLE

nums = (2, 3, 1, 9, 3, 5)

# nums.sort() # AttributeError: tuple doesn't support sort() method

print(nums)

## # SORTING DICTIONARY

employee = {'name':'John', 'age':26, 'lang':['Python', 'Java']}

employee.sort() # AttributeError: dict does n't support sort() method

emp\_s = sorted(employee)

print(emp\_s) # ['age', 'lang', 'name']

# 2nd e.g

class Employee:

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

def \_\_repr\_\_(self):

return '({}, {}, {})'.format(self.name, self.age, self.salary)

e1 = Employee("Carl", 37, 70000)

e2 = Employee("Sarah", 29, 80000)

e3 = Employee("John", 43, 90000)

employees = [e1, e2, e3]

def emp\_sort(emp):

return emp.name

s\_employees = sorted(employees, key=emp\_sort, reverse=True) # or below

# s\_employees = sorted(employees, key= lambda emp: emp.name, reverse=True)

# above is identical to the one below

# from operator import attrgetter

# s\_employees = sorted(employees, key=attrgetter("age") , reverse=True)

print(s\_employees)

# [(Sarah, 29, 80000), (John, 43, 90000), (Carl, 37, 70000)]