Important Points

List

**List** is mutable i.e. you can change the values on list anytime you want.

Python List Methods

**append()** - Add an element to the end of the list

lst.append(9)

**extend()** - Add all elements of a list to the another list

lst.extend([1,2,3])

**insert()** - Insert an item at the defined index

lst.insert(2,3)

**del listname [index]** – Deletes the whole list or elements specified

del lst [2:4]

del lst

**remove()** - Removes an item from the list

lst.remove(‘p’)

**pop(index)** - Removes and returns an element at the given index, if index is not given removes the last element from the list.

lst.pop(1)

lst.pop()

**clear()** - Removes all items from the list

lst.clear()

**index()** - Returns the index of the first matched item

lst.index(‘a’)

**count()** - Returns the count of number of items passed as an argument

lst.count(‘a’)

**sort()** - Sort items in a list in ascending order – lst.sort()

**reverse()** - Reverse the order of items in the list – lst.reverse()

**copy()** - Returns a shallow copy of the list. In case of list1 = list2 assignment the copy of the list is created, but whenever we change any list the other also changes. To avoid this we need to use the copy function.

List2 = List1.copy()

Tuple

**Tuple** is immutable i.e. you cannot change the values in Tuple in runtime.

For example, List is used when you should store the share prices which changes daily, and Tuple is used for storing the days of week or months of year.

Tuple is immutable but if you have mutable element(list) in Tuple then it can be changed.

E.g.

>>> my\_tup = (1,2,[3,4,5])

>>> my\_tup [0] = 0

TypeError: 'tuple' object does not support item assignment

>>> my\_tup [2][2] = 6

>>> my\_tup

(1,2,[3,4,6]

Python Tuple methods:

**count(x) -** Return the number of items that is equal to x

**index(x) -** Return index of first item that is equal to x

We can use the sorted(tuple, reverser = True/False) function to get the elements in sorted order but it won’t change the actual tuple.

Dictionary

**Dictionary** is used to store the data in key value format, for example storing country and its capital.

**clear()** Remove all items form the dictionary.

**copy()** Return a shallow copy of the dictionary.

**fromkeys(seq[, v])** - Return a new dictionary with keys from seq and value equal to v (defaults to None).

**get(key[,d])** - Return the value of key. If key doesnot exit, return d (defaults to None).

**items()** - Return a new view of the dictionary's items (key, value).

**keys()** - Return a new view of the dictionary's keys.

**pop(key[,d]) -** Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.

**popitem()** - Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.

**setdefault(key[,d])** - If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).

**update([other]) -** Update the dictionary with the key/value pairs from other dictionary, overwriting existing keys.

**values() -** Return a new view of the dictionary's values

Different ways to create the dictionary:

1. Di = {‘Name’:’Abhay’, ‘Sal’:120000}
2. Di = dict(Name = ‘Abhay’, Sal = 120000)
3. Kys = (‘Name’, ‘Sal’)

Vals = (‘Abhay’, 120000)

Di = dict(zip(Kys,Vals))

1. Kys = (‘Name’, ‘Sal’)

Di = dict.fromkeys(Kys,’?’)

Print (Di) -> {'Name': '?', 'Age': '?'}

Set

**Set** is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

In Set {} we can remove the element from set using remove () and discard () functions

Only diff between two commands is remove will raise error if the element is not present in the set.

>>>setName.discard(5)

>>>setName.remove(5)

*KeyError: 5*

**add()** Adds an element to the set

**clear()** Removes all elements from the set

**copy()** Returns a copy of the set

**difference()** Returns the difference of two or more sets as a new set

**difference\_update()** Removes all elements of another set from this set

**discard()** Removes an element from the set if it is a member. (Do nothing if the element is not in set)

**intersection()** Returns the intersection of two sets as a new set

**intersection\_update()** Updates the set with the intersection of itself and another

**isdisjoint()** Returns True if two sets have a null intersection

**issubset()** Returns True if another set contains this set

**issuperset()** Returns True if this set contains another set

**pop()** Removes and returns an arbitary set element. Raise KeyError if the set is empty

**remove()** Removes an element from the set. If the element is not a member, raise a KeyError

**symmetric\_difference()** Returns the symmetric difference of two sets as a new set

**symmetric\_difference\_update()** Updates a set with the symmetric difference of itself and another

**union()** Returns the union of sets in a new set

**update()** Updates the set with the union of itself and others

**Important built-in functions**

* isdigit() -> a.isdigit()
* isalpha() -> a.isalpha()
* isupper() -> a.isupper()
* islower() -> a.islower()
* int(x,2) -> To convert x- binary number to decimal number
* sorted (list, reverse = True/False(default), key=fuctionName) -> To get the new sorted list, list.sort(reverse=True/False(default)) will sort the existing list itself.

FuncationName can be inbuild or user defined function or lambda function can be used.

* set(list) -> to convert list into set
* tuple(list) -> To convert list into tuple
* math.sqrt(x) -> To get square root of number. Math is library first need to import it.
* upper () -> string.upper()
* strip () -> string.strip() To trim the spaces from string
* re.search("[a-z]",txt)
* re.search ("[0-9]",txt)
* re.search ("[A-Z]"
* help (class name) -> This command will give information on any class, user defined or system defined.
* setattr -> This function is used to add new attribute to the class
* delattr -> This function is used to delete the attribute from the class.
* Setattr -> This function is used to get the attribute value

class Temp1:

testvar = 100

t = Temp1()

print(t.testvar)

setattr(t,'testvar',200)

setattr(t,'testnum','xyz')

print(t.testvar)

print(t.testnum)

print(t.\_\_dict\_\_)

delattr(t,'testnum')

del t.testvar

print(t.\_\_dict\_\_)

print(getattr(t,'testvar'))

Output :

100

200

xyz

{'testvar': 200, 'testnum': 'xyz'}

{}

Random Module

from random import \*

**random()**: This function always generate some float value between 0 and 1 ( not inclusive)

0.8413487247317323

0.9524576453805842

0.8883838878162098

0.040135854947829785

0.458988673691354

**randint**(start, stop): To generate random integer between two given numbers(inclusive)

randint(20,30)

27

30

23

23

28

**uniform**(start, stop) : It returns random float values between 2 given numbers(not inclusive)

uniform(20,30)

26.985961179286637

24.242237175452765

22.07294639788659

20.697337031211397

26.366607225048888

**randrange**([start],stop,[step]) : start and step is optional with default values 0 and 1 respectively

randrange(1,11,2)

5

5

1

7

3

**choice**([list]): Picks the random object from the given list

Function Decorators

Decorator is a function which takes the function as argument and extends its functionality and returns the modified function with extended functionality without changing the argument function.

def decor(func):

def inner(name):

if name == 'xxx':

print('Hello',name,'bad morning')

else:

func(name)

return inner

@decor

def wish(name):

print ('Hello',name,'Good morning')

>>> wish('xxx')

Hello xxx bad morning

>>>wish('abc')

Hello abc Good morning

*If we modify the decorator function it will not show the modified functionality in the argument function. We need to define the argument function (in above case wish function) again.*

For connecting the database:



Global, local and nonlocal variables

x = 999

def outer():

x = 10

def inner():

nonlocal x

x = 30

print('Inner x ',x)

inner()

print('Outer x ',x)

print('Before global x ',x)

outer()

print('Global x ',x)

Global variable scope is available for whole session

Local variable scope is for the function/method or its child functions/methods

Nonlocal variable scope is within the parent and child function/method

Before global x 999

Inner x 30

Outer x 30

Global x 999