**Concept collection**

List/tuple/Dictionary Operations

listToStr = ','.join(ListName)

converting list[A, B ,C, D] into string comma seprated like A,B,C,D --

List = list(dict.fromkeys(List)) # To remove list redundancy

Citem =(sorted(Unsorted,key=lambda x:(x[1],x[0])))

# sorting list of tuple Unsorted by both the elements.

List.count(i) # To get element i count in list List

LF = [x for x in LF if x]

# removing blank list from Nested List LF

Strings Operations

String.split(',') # To convert a string into list comma sepearated.

list(input())

# converting a input string into character sererated List like : Ai32==>['A','i',3,2]

x =1

print(eval('x+4+2\*\*x'))

eval() bilt-in function parse the input string as python expression

and assign the value to variable if passed

Special Symbols

def fn(\*arg):

    for ar in arg:

        print(ar)

fn('A',123, '456' ,'ccd', 556 ,91)

#  \*args (Non-Keyword mutiple Arguments) , can be list or tuple

def fn(\*\*karg):

    for argkey,argv in karg.items():

        print(argkey ," >>> ",argv)

fn(A='123', H ='ccd', ccd =91)

#  \*\*karg (Keyword mutiple Arguments) , can be dictionary

Pandas Dataframe

df\_csvR = pd.read\_csv("BookRecords.csv",delimiter=',' ) # Reading csv file and storing in a dataframe "df\_csvR"

D = dict(zip(df\_csvR.B\_Name, df\_csvR.B\_Genre)) # Converting Dataframe df\_csvR columns into Dictionary key,value pair

L = df\_csvR.B\_Genre.tolist()

# Converting dataframe df\_csvR column B\_Genre to list

df\_csvR[df\_csvR['B\_Genre'].str.contains(substring ,flags=re.IGNORECASE, regex=True)]['B\_Name']

# To find substring of Dataframe df\_csvR in any column like 'B\_Genre'

with flag to Ignore case-senstivity and return a result

column 'B\_Name' rows for matching data.

Decorators

Very powerful and useful tool -- allows programmers to modify/enhance the behaviour and properties of any function or class by wrapping it inside another function , so that the original function won’t get modified permanently -- there are some predefined and one can also create there custom decorators.

import time

from pymongo import MongoClient

def ExecutionTime(func):

    '''

    This decorator is to calculate execution time of any function

    # calling decorator with a function

    @ExecutionTime

    def fun\_mongoconnect(CollectionName):

        ---- fun body ----

    '''

    def innerMethod(\*args,\*\*kwargs):

        Execution\_begin = time.time()

        res = func(\*args ,\*\*kwargs)

        Execution\_end = time.time()

        print("Total time taken by function ({0}) is {1} sec : ".format(func.\_\_name\_\_, round(Execution\_end - Execution\_begin,4)))

    return innerMethod