1. **Python | Convert a list of Tuples into Dictionary**

# Python code to convert into dictionary

def Convert(tup, di):

  for a, b in tup:

    di.setdefault(a, []).append(b)

  return di

# Driver Code

tups = [("akash", 10), ("gaurav", 12), ("anand", 14),

  ("suraj", 20), ("akhil", 25), ("ashish", 30)]

dictionary = {}

print (Convert(tups, dictionary))

**Output:** {'akash': [10], 'gaurav': [12], 'anand': [14], 'suraj': [20], 'akhil': [25], 'ashish': [30]}

1. **Python counter and dictionary intersection example (Make a string using deletion and rearrangement)**

# Python code to find if we can make first string from second by deleting some characters from second and rearranging remaining characters.

from collections import Counter

def makeString(str1,str2):

  # convert both strings into dictionaries

  # output will be like str1="aabbcc",

  # dict1={'a':2,'b':2,'c':2}

  # str2 = 'abbbcc', dict2={'a':1,'b':3,'c':2}

  dict1 = Counter(str1)

  dict2 = Counter(str2)

  # take intersection of two dictionaries

  # output will be result = {'a':1,'b':2,'c':2}

  result = dict1 & dict2

  # compare resultant dictionary with first

  # dictionary comparison first compares keys

  # and then compares their corresponding values

  return result == dict1

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

  str1 = 'ABHISHEKsinGH'

  str2 = 'gfhfBHkooIHnfndSHEKsiAnG'

  if (makeString(str1,str2)==True):

    print("Possible")

  else:

    print("Not Possible")

**Output:** Possible

1. **Python dictionary, set and counter to check if frequencies can become same**

# Function to Check if frequency of all characters can become same by one removal

from collections import Counter

def allSame(input):

  # calculate frequency of each character

  # and convert string into dictionary

  dict=Counter(input)

  # now get list of all values and push it

  # in set

  same = list(set(dict.values()))

  if len(same)>2:

    print('No')

  elif len (same)==2 and same[1]-same[0]>1:

    print('No')

  else:

    print('Yes')

  # now check if frequency of all characters

  # can become same

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

  input = 'xxxyyzzt'

  allSame(input)

**Output:** No

1. **Scraping And Finding Ordered Words In A Dictionary using Python**

# Python program to find ordered words

import requests

# Scrapes the words from the URL below and stores

# them in a list

def getWords():

  # contains about 2500 words

  url = "http://www.puzzlers.org/pub/wordlists/unixdict.txt"

  fetchData = requests.get(url)

  # extracts the content of the webpage

  wordList = fetchData.content

  # decodes the UTF-8 encoded text and splits the

  # string to turn it into a list of words

  wordList = wordList.decode("utf-8").split()

  return wordList

# function to determine whether a word is ordered or not

def isOrdered():

  # fetching the wordList

  collection = getWords()

  # since the first few of the elements of the

  # dictionary are numbers, getting rid of those

  # numbers by slicing off the first 17 elements

  collection = collection[16:]

  word = ''

  for word in collection:

    result = 'Word is ordered'

    i = 0

    l = len(word) - 1

    if (len(word) < 3): # skips the 1 and 2 lettered strings

      continue

    # traverses through all characters of the word in pairs

    while i < l:

      if (ord(word[i]) > ord(word[i+1])):

        result = 'Word is not ordered'

        break

      else:

        i += 1

    # only printing the ordered words

    if (result == 'Word is ordered'):

      print(word,': ',result)

# execute isOrdered() function

if \_\_name\_\_ == '\_\_main\_\_':

  isOrdered()

**Output:** aau: Word is ordered

abbe: Word is ordered

abbey: Word is ordered

abbot: Word is ordered

abbott: Word is ordered

abc: Word is ordered

abe: Word is ordered

abel: Word is ordered

abet: Word is ordered

abo: Word is ordered

abort: Word is ordered

accent: Word is ordered

accept: Word is ordered

1. **Possible Words using given characters in Python**

# Function to print words which can be created

# using given set of characters

def charCount(word):

  dict = {}

  for i in word:

    dict[i] = dict.get(i, 0) + 1

  return dict

def possible\_words(lwords, charSet):

  for word in lwords:

    flag = 1

    chars = charCount(word)

    for key in chars:

      if key not in charSet:

        flag = 0

      else:

        if charSet.count(key) != chars[key]:

          flag = 0

    if flag == 1:

      print(word)

if \_\_name\_\_ == "\_\_main\_\_":

  input = ['goo', 'bat', 'me', 'eat', 'goal', 'boy', 'run']

  charSet = ['e', 'o', 'b', 'a', 'm', 'g', 'l']

  possible\_words(input, charSet)

**Output:** me

goal

1. **Python – Keys associated with Values in Dictionary**

# Python3 code to demonstrate working of

# Value's Key association

# Using loop + items()

# initializing dictionary

test\_dict = {'gfg' : [4, 5], 'is' : [8], 'best' : [10, 12]}

# printing original dictionary

print("The original dictionary : " + str(test\_dict))

# initializing value list

val\_list = [5, 10]

# Value's Key association

# Using loop + items()

temp = {}

for key, vals in test\_dict.items():

  for val in vals:

    temp[val] = key

res = [temp[ele] for ele in val\_list]

# printing result

print("The keys mapped to " + str(val\_list) + " are : " + str(res))

**Output:** The original dictionary : {'gfg': [4, 5], 'is': [8], 'best': [10, 12]}

The keys mapped to [5, 10] are : ['gfg', 'best']

1. **Python program to Find the size of a Tuple**

import sys

# sample Tuples

Tuple1 = ("A", 1, "B", 2, "C", 3)

Tuple2 = ("Geek1", "Raju", "Geek2", "Nikhil", "Geek3", "Deepanshu")

Tuple3 = ((1, "Lion"), ( 2, "Tiger"), (3, "Fox"), (4, "Wolf"))

# print the sizes of sample Tuples

print("Size of Tuple1: " + str(sys.getsizeof(Tuple1)) + "bytes")

print("Size of Tuple2: " + str(sys.getsizeof(Tuple2)) + "bytes")

print("Size of Tuple3: " + str(sys.getsizeof(Tuple3)) + "bytes")

**Output:** Size of Tuple1: 88bytes

Size of Tuple2: 88bytes

Size of Tuple3: 72bytes

1. **Python – Maximum and Minimum K elements in Tuple**

# Python3 code to demonstrate working of

# Maximum and Minimum K elements in Tuple

# Using sorted() + loop

# initializing tuple

test\_tup = (5, 20, 3, 7, 6, 8)

# printing original tuple

print("The original tuple is : " + str(test\_tup))

# initializing K

K = 2

# Maximum and Minimum K elements in Tuple

# Using sorted() + loop

res = []

test\_tup = list(sorted(test\_tup))

for idx, val in enumerate(test\_tup):

  if idx < K or idx >= len(test\_tup) - K:

    res.append(val)

res = tuple(res)

# printing result

print("The extracted values : " + str(res))

**Output:** The original tuple is : (5, 20, 3, 7, 6, 8)

The extracted values : (3, 5, 8, 20)

1. **Create a list of tuples from given list having number and its cube in each tuple**

# Python program to create a list of tuples

# from given list having number and

# its cube in each tuple

# creating a list

list1 = [1, 2, 5, 6]

# using list comprehension to iterate each

# values in list and create a tuple as specified

res = [(val, pow(val, 3)) for val in list1]

# print the result

print(res)

**Output:** [(1, 1), (2, 8), (5, 125), (6, 216)]

1. **Python – Adding Tuple to List and vice – versa**

# Python3 code to demonstrate working of

# Adding Tuple to List and vice - versa

# Using += operator (list + tuple)

# initializing list

test\_list = [5, 6, 7]

# printing original list

print("The original list is : " + str(test\_list))

# initializing tuple

test\_tup = (9, 10)

# Adding Tuple to List and vice - versa

# Using += operator (list + tuple)

test\_list += test\_tup

# printing result

print("The container after addition : " + str(test\_list))

**Output:** The original list is : [5, 6, 7]

The container after addition : [5, 6, 7, 9, 10]

1. **Python – Closest Pair to Kth index element in Tuple**

# Python3 code to demonstrate working of

# Closest Pair to Kth index element in Tuple

# Using enumerate() + loop

# initializing list

test\_list = [(3, 4), (78, 76), (2, 3), (9, 8), (19, 23)]

# printing original list

print("The original list is : " + str(test\_list))

# initializing tuple

tup = (17, 23)

# initializing K

K = 1

# Closest Pair to Kth index element in Tuple

# Using enumerate() + loop

min\_dif, res = 999999999, None

for idx, val in enumerate(test\_list):

  dif = abs(tup[K - 1] - val[K - 1])

  if dif < min\_dif:

    min\_dif, res = dif, idx

# printing result

print("The nearest tuple to Kth index element is : " + str(test\_list[res]))

**Output:** The original list is : [(3, 4), (78, 76), (2, 3), (9, 8), (19, 23)]

The nearest tuple to Kth index element is : (19, 23)