1. **Python – Sort Dictionary key and values List**

# Python3 code to demonstrate working of

# Sort Dictionary key and values List

# Using loop + dictionary comprehension

# initializing dictionary

test\_dict = {'gfg': [7, 6, 3],

      'is': [2, 10, 3],

      'best': [19, 4]}

# printing original dictionary

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

# Sort Dictionary key and values List

# Using loop + dictionary comprehension

res = dict()

for key in sorted(test\_dict):

  res[key] = sorted(test\_dict[key])

# printing result

print("The sorted dictionary : " + str(res))

**Output:** The original dictionary is : {'gfg': [7, 6, 3], 'is': [2, 10, 3], 'best': [19, 4]}

The sorted dictionary : {'best': [4, 19], 'gfg': [3, 6, 7], 'is': [2, 3, 10]}

1. **Handling missing keys in Python dictionaries**

# Python code to demonstrate Dictionary and missing value error

# initializing Dictionary

d = { 'a' : 1 , 'b' : 2 }

# trying to output value of absent key

print ("The value associated with 'c' is : ", d['c'])

**Output:** KeyError Traceback (most recent call last)

[<ipython-input-8-bb278dec5812>](https://localhost:8080/) in <module>

**6**

**7** # trying to output value of absent key

----> 8 print ("The value associated with 'c' is : ", d['c'])

KeyError: 'c'

1. **Python dictionary with keys having multiple inputs**

# Python code to demonstrate a dictionary

# with multiple inputs in a key.

import random as rn

# creating an empty dictionary

dict = {}

# Insert first triplet in dictionary

x, y, z = 10, 20, 30

dict[x, y, z] = x + y - z;

# Insert second triplet in dictionary

x, y, z = 5, 2, 4

dict[x, y, z] = x + y - z;

# print the dictionary

print(dict)

**Output:** {(10, 20, 30): 0, (5, 2, 4): 3}

1. **Print anagrams together in Python using List and Dictionary**

# Function to return all anagrams together

def allAnagram(input):

  # empty dictionary which holds subsets

  # of all anagrams together

  dict = {}

  # traverse list of strings

  for strVal in input:

    # sorted(iterable) method accepts any

    # iterable and returns list of items

    # in ascending order

    key = ''.join(sorted(strVal))

    # now check if key exist in dictionary

    # or not. If yes then simply append the

    # strVal into the list of it's corresponding

    # key. If not then map empty list onto

    # key and then start appending values

    if key in dict.keys():

      dict[key].append(strVal)

    else:

      dict[key] = []

      dict[key].append(strVal)

  # traverse dictionary and concatenate values

  # of keys together

  output = ""

  for key,value in dict.items():

    output = output + ' '.join(value) + ' '

  return output

# Driver function

if \_\_name\_\_ == "\_\_main\_\_":

  input=['cat', 'dog', 'tac', 'god', 'act']

  print (allAnagram(input))

**Output:** cat tac act dog god

1. **K’th Non-repeating Character in Python using List Comprehension and OrderedDict**

# Function to find k'th non repeating character

# in string

from collections import OrderedDict

def kthRepeating(input,k):

  # OrderedDict returns a dictionary data

    # structure having characters of input

  # string as keys in the same order they

    # were inserted and 0 as their default value

  dict=OrderedDict.fromkeys(input,0)

  # now traverse input string to calculate

    # frequency of each character

  for ch in input:

    dict[ch]+=1

  # now extract list of all keys whose value

    # is 1 from dict Ordered Dictionary

  nonRepeatDict = [key for (key,value) in dict.items() if value==1]

  # now return (k-1)th character from above list

  if len(nonRepeatDict) < k:

    return 'Less than k non-repeating characters in input.'

  else:

    return nonRepeatDict[k-1]

# Driver function

if \_\_name\_\_ == "\_\_main\_\_":

  input = "geeksforgeeks"

  k = 3

  print (kthRepeating(input, k))

**Output:** r

1. **Check if binary representations of two numbers are anagram**

# function to Check if binary representations

# of two numbers are anagram

from collections import Counter

def checkAnagram(num1,num2):

  # convert numbers into in binary

  # and remove first two characters of

  # output string because bin function

  # '0b' as prefix in output string

  bin1 = bin(num1)[2:]

  bin2 = bin(num2)[2:]

  # append zeros in shorter string

  zeros = abs(len(bin1)-len(bin2))

  if (len(bin1)>len(bin2)):

    bin2 = zeros \* '0' + bin2

  else:

    bin1 = zeros \* '0' + bin1

  # convert binary representations

  # into dictionary

  dict1 = Counter(bin1)

  dict2 = Counter(bin2)

  # compare both dictionaries

  if dict1 == dict2:

    print('Yes')

  else:

    print('No')

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

  num1 = 8

  num2 = 4

  checkAnagram(num1,num2)

**Output:** Yes

1. **Python Counter to find the size of largest subset of anagram words**

# Function to find the size of largest subset of anagram words

from collections import Counter

def maxAnagramSize(input):

  # split input string separated by space

  input = input.split(" ")

  # sort each string in given list of strings

  for i in range(0,len(input)):

    input[i]=''.join(sorted(input[i]))

  # now create dictionary using counter method

  # which will have strings as key and their

  # frequencies as value

  freqDict = Counter(input)

  # get maximum value of frequency

  print (max(freqDict.values()))

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

  input = 'ant magenta magnate tan gnamate'

  maxAnagramSize(input)

**Output:** 3

1. **Python | Remove all duplicates words from a given sentence**

from collections import Counter

def remov\_duplicates(input):

  # split input string separated by space

  input = input.split(" ")

  # now create dictionary using counter method

  # which will have strings as key and their

  # frequencies as value

  UniqW = Counter(input)

  # joins two adjacent elements in iterable way

  s = " ".join(UniqW.keys())

  print (s)

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

  input = 'Python is great and Java is also great'

  remov\_duplicates(input)

**Output:** Python is great and Java also

1. **Python Dictionary to find mirror characters in a string**

# function to mirror characters of a string

def mirrorChars(input,k):

  # create dictionary

  original = 'abcdefghijklmnopqrstuvwxyz'

  reverse = 'zyxwvutsrqponmlkjihgfedcba'

  dictChars = dict(zip(original,reverse))

  # separate out string after length k to change

  # characters in mirror

  prefix = input[0:k-1]

  suffix = input[k-1:]

  mirror = ''

  # change into mirror

  for i in range(0,len(suffix)):

    mirror = mirror + dictChars[suffix[i]]

  # concat prefix and mirrored part

  print (prefix+mirror)

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

  input = 'paradox'

  k = 3

  mirrorChars(input,k)

**Output:** paizwlc

1. **Counting the frequencies in a list using dictionary in Python**

# Python program to count the frequency of

# elements in a list using a dictionary

def CountFrequency(my\_list):

  # Creating an empty dictionary

  freq = {}

  for item in my\_list:

    if (item in freq):

      freq[item] += 1

    else:

      freq[item] = 1

  for key, value in freq.items():

    print ("% d : % d"%(key, value))

# Driver function

if \_\_name\_\_ == "\_\_main\_\_":

  my\_list =[1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2, 2]

  CountFrequency(my\_list)

**Output:** 1 : 5

5 : 2

3 : 3

4 : 3

2 : 4