1. **Python – Join Tuples if similar initial element**

# Python3 code to demonstrate working of

# Join Tuples if similar initial element

# Using loop

# initializing list

test\_list = [(5, 6), (5, 7), (6, 8), (6, 10), (7, 13)]

# printing original list

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

# Join Tuples if similar initial element

# Using loop

res = []

for sub in test\_list:

  if res and res[-1][0] == sub[0]:

    res[-1].extend(sub[1:])

  else:

    res.append([ele for ele in sub])

res = list(map(tuple, res))

# printing result

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

**Output:** The original list is : [(5, 6), (5, 7), (6, 8), (6, 10), (7, 13)]

The extracted elements : [(5, 6, 7), (6, 8, 10), (7, 13)]

1. **Python – Extract digits from Tuple list**

# Python3 code to demonstrate working of

# Extract digits from Tuple list

# Using map() + chain.from\_iterable() + set() + loop

from itertools import chain

# initializing list

test\_list = [(15, 3), (3, 9), (1, 10), (99, 2)]

# printing original list

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

# Extract digits from Tuple list

# Using map() + chain.from\_iterable() + set() + loop

temp = map(lambda ele: str(ele), chain.from\_iterable(test\_list))

res = set()

for sub in temp:

  for ele in sub:

    res.add(ele)

# printing result

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

**Output:** The original list is : [(15, 3), (3, 9), (1, 10), (99, 2)]

The extracted digits : {'0', '9', '1', '3', '2', '5'}

1. **Python – All pair combinations of 2 tuples**

# Python3 code to demonstrate working of

# All pair combinations of 2 tuples

# Using list comprehension

# initializing tuples

test\_tuple1 = (4, 5)

test\_tuple2 = (7, 8)

# printing original tuples

print("The original tuple 1 : " + str(test\_tuple1))

print("The original tuple 2 : " + str(test\_tuple2))

# All pair combinations of 2 tuples

# Using list comprehension

res = [(a, b) for a in test\_tuple1 for b in test\_tuple2]

res = res + [(a, b) for a in test\_tuple2 for b in test\_tuple1]

# printing result

print("The filtered tuple : " + str(res))

**Output:** The original tuple 1 : (4, 5)

The original tuple 2 : (7, 8)

The filtered tuple : [(4, 7), (4, 8), (5, 7), (5, 8), (7, 4), (7, 5), (8, 4), (8, 5)]

1. **Python – Remove Tuples of Length K**

# Python3 code to demonstrate working of

# Remove Tuples of Length K

# Using list comprehension

# initializing list

test\_list = [(4, 5), (4, ), (8, 6, 7), (1, ), (3, 4, 6, 7)]

# printing original list

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

# initializing K

K = 1

# 1 liner to perform task

# filter just lengths other than K

# len() used to compute length

res = [ele for ele in test\_list if len(ele) != K]

# printing result

print("Filtered list : " + str(res))

**Output:** The original list : [(4, 5), (4,), (8, 6, 7), (1,), (3, 4, 6, 7)]

Filtered list : [(4, 5), (8, 6, 7), (3, 4, 6, 7)]

1. **Sort a list of tuples by second Item**

# Python program to sort a list of tuples by the second Item

# Function to sort the list of tuples by its second item

def Sort\_Tuple(tup):

  # getting length of list of tuples

  lst = len(tup)

  for i in range(0, lst):

    for j in range(0, lst-i-1):

      if (tup[j][1] > tup[j + 1][1]):

        temp = tup[j]

        tup[j] = tup[j + 1]

        tup[j + 1] = temp

  return tup

# Driver Code

tup = [('for', 24), ('is', 10), ('Geeks', 28),

  ('Geeksforgeeks', 5), ('portal', 20), ('a', 15)]

print(Sort\_Tuple(tup))

**Output:** [('Geeksforgeeks', 5), ('is', 10), ('a', 15), ('portal', 20), ('for', 24), ('Geeks', 28)]

1. **Python program to Order Tuples using external List**

my\_list = [('Mark', 34), ('Will', 91), ('Rob', 23)]

print("The list of tuple is : ")

print(my\_list)

ordered\_list = ['Will', 'Mark', 'Rob']

print("The ordered list is :")

print(ordered\_list)

temp = dict(my\_list)

my\_result = [(key, temp[key]) for key in ordered\_list]

print("The ordered tuple list is : ")

print(my\_result)

**Output:** The list of tuple is :

[('Mark', 34), ('Will', 91), ('Rob', 23)]

The ordered list is :

['Will', 'Mark', 'Rob']

The ordered tuple list is :

[('Will', 91), ('Mark', 34), ('Rob', 23)

1. **Python – Flatten tuple of List to tuple**

# Python3 code to demonstrate working of

# Flatten tuple of List to tuple

# Using sum() + tuple()

# initializing tuple

test\_tuple = ([5, 6], [6, 7, 8, 9], [3])

# printing original tuple

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

# Flatten tuple of List to tuple

# Using sum() + tuple()

res = tuple(sum(test\_tuple, []))

# printing result

print("The flattened tuple : " + str(res))

**Output:** The original tuple : ([5, 6], [6, 7, 8, 9], [3])

The flattened tuple : (5, 6, 6, 7, 8, 9, 3)

1. **Python – Convert Nested Tuple to Custom Key Dictionary**

# Python3 code to demonstrate working of

# Convert Nested Tuple to Custom Key Dictionary

# Using list comprehension + dictionary comprehension

# initializing tuple

test\_tuple = ((4, 'Gfg', 10), (3, 'is', 8), (6, 'Best', 10))

# printing original tuple

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

# Convert Nested Tuple to Custom Key Dictionary

# Using list comprehension + dictionary comprehension

res = [{'key': sub[0], 'value': sub[1], 'id': sub[2]}

              for sub in test\_tuple]

# printing result

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

**Output:** The original tuple : ((4, 'Gfg', 10), (3, 'is', 8), (6, 'Best', 10))

The converted dictionary : [{'key': 4, 'value': 'Gfg', 'id': 10}, {'key': 3, 'value': 'is', 'id': 8}, {'key': 6, 'value': 'Best', 'id': 10}]

1. **Python Program for Binary Search (Recursive and Iterative)**

# Python 3 program for recursive binary search.

# Modifications needed for the older Python 2 are found in comments.

# Returns index of x in arr if present, else -1

def binary\_search(arr, low, high, x):

  # Check base case

  if high >= low:

    mid = (high + low) // 2

    # If element is present at the middle itself

    if arr[mid] == x:

      return mid

    # If element is smaller than mid, then it can only

    # be present in left subarray

    elif arr[mid] > x:

      return binary\_search(arr, low, mid - 1, x)

    # Else the element can only be present in right subarray

    else:

      return binary\_search(arr, mid + 1, high, x)

  else:

    # Element is not present in the array

    return -1

# Test array

arr = [ 2, 3, 4, 10, 40 ]

x = 10

# Function call

result = binary\_search(arr, 0, len(arr)-1, x)

if result != -1:

  print("Element is present at index", str(result))

else:

  print("Element is not present in array")

**Output:** Element is present at index 3

1. **Python Program for Linear Search**

# Searching an element in a list/array in python

# can be simply done using \'in\' operator

# Example:

# if x in arr:

# print arr.index(x)

# If you want to implement Linear Search in python

# Linearly search x in arr[]

# If x is present then return its location

# else return -1

def search(arr, x):

  for i in range(len(arr)):

    if arr[i] == x:

      return i

  return -1

1. **Python Program for Insertion Sort**

# Python program for implementation of Insertion Sort

# Function to do insertion sort

def insertionSort(arr):

  if (n := len(arr)) <= 1:

    return

  for i in range(1, n):

    key = arr[i]

    # Move elements of arr[0..i-1], that are

    # greater than key, to one position ahead

    # of their current position

    j = i-1

    while j >=0 and key < arr[j] :

      arr[j+1] = arr[j]

      j -= 1

    arr[j+1] = key

#sorting the array [12, 11, 13, 5, 6] using insertionSort

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

print(arr)

**Output:** [5, 6, 11, 12, 13]