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

```
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))
```

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

```
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))
```

3. **Python – All pair combinations of 2 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))
```

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

```
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))
```

5. **Sort a list of tuples by 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))
```

6. **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)
```

7. **Python – Flatten tuple of List to 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))
```

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

```
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))
```

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

```
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")
```

10. Python Program for Linear Search

```
def search(arr, x):

        for i in range(len(arr)):

                if arr[i] == x:
                        return i

        return -1
```

11. Python Program for Insertion Sort

```
def insertionSort(arr):


        # Traverse through 1 to len(arr)
        for i in range(1, len(arr)):


                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
# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print ("Sorted array is:")
for i in range(len(arr)):
        print ("%d" %arr[i])
```