**Week-4:**

1. Write a function called is\_sorted that takes a list as a parameter and returns True if the list is sorted in ascending order and False otherwise.

def is\_sorted(a):

for i in range(len(a)-1):

if a[i]>=a[i+1]:

return False

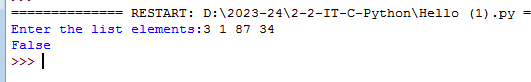
return True

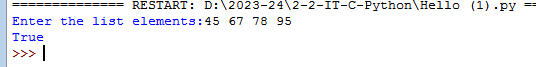
input\_string=input("Enter the list elements:")

vlist= input\_string.split()

print(is\_sorted(vlist))

**Output:**





1. Write a function called has\_duplicates that takes a list and returns True if there is any element that appears more than once. It should not modify the original list.

def has\_duplicates(vlist):

n=len(vlist)

rep=[]

for i in range(n):

k=i+1

for j in range(k,n):

if vlist[i]==vlist[j]:

return 'True'

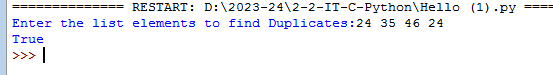
return 'False'

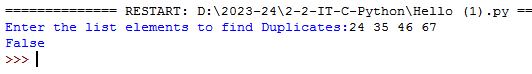
input\_string=input("Enter the list elements:")

vlist= input\_string.split()

print(has\_duplicates(vlist))

**Output:**





1. Write a function called remove\_duplicates that takes a list and returns a new list with

only the unique elements from the original. *Hint: they don’t have to be in the same   
order.*

def remove\_duplicates(vlist):

final\_list = []

for num in vlist:

if num not in final\_list:

final\_list.append(num)

return final\_list

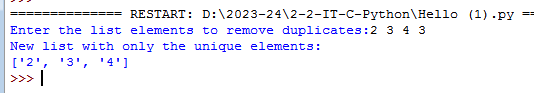
input\_string=input("Enter the list elements:")

vlist= input\_string.split()

print(“New list with only the unique elements:”)

print(remove\_duplicates(vlist))

**Output:**



1. The wordlist I provided, words.txt, doesn’t contain single letter words. So you might want to add “I”, “a”, and the empty string.

def reduce\_words():

def create\_dict():

         fin = open('words.txt')

         res = {}

         for line in fin:

             word = line.strip()

             res[word] = []

         for element in ["a", "i", ""]:

             res[element] = []

         return res

     def add\_children(d):

         for key in d.keys():

             children = []

              for i in range(len(key)):

                 candidate = key[:i] + key[i+1:]

                 if candidate in d and candidate not in children:

                     children.append(candidate)

              d[key] = children

         return d

     def recursive\_trails(d):

         res = []

         def helper(key, result):

             if d[key] == []:

                 return

             if key in ["a", "i", ""]:

                 res.append((len(result[0]), result))

             else:

                 for entry in d[key]:

                     return helper(entry, result + [entry])

         for key,value in d.items():

             helper(key, [key])

         return res

     def top\_n(results, n):

         results.sort(reverse = True)

         for i in range(n):

             print results[i]

     d = create\_dict()

     add\_children(d)

     trails = recursive\_trails(d)

     top\_n(trails, 20)

reduce\_words()

1. Write a python code to read a dictionary values from the user. Construct a function to invert its content. i.e., keys should be values and values should be keys.

ini\_dict = {101: "akshat", 201 : "ball"}

# print initial dictionary

print("initial dictionary : ", str(ini\_dict))

# inverse mapping using dict comprehension

inv\_dict = {v: k for k, v in ini\_dict.items()}

# print final dictionary

print("inverse mapped dictionary : ", str(inv\_dict))

**Output:**

