ASSIGNMENT – 2 DAA0666

11. Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the container contains the most water.
Return *the maximum amount of water a container can store*. Notice that you may not slant the container.

CODE:

```
def maxArea(A,

Len) :

area = 0 for i in range(Len) : for j in

range(i + 1, Len) : # Calculating the

max area area = max(area, min(A[j],

A[i]) * (j - i)) return area # Driver

code

a = [ 1, 5, 4, 3 ] b = [

3, 1, 2, 4, 5 ] len1 =

len(a)

print(maxArea(a,

len1)) len2 = len(b)

print(maxArea(b,

len2))
```

OUTPUT:

12. Roman numerals are represented by seven different symbols: I, V, X, L, C, D and I **2**
V **5**
X **5**

M.
Symbol Value
L 50
C 100
D 500
M 1000
For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as
XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.
Roman numerals are usually written largest to smallest from left to right. However, the numeral
for four is not IIII. Instead, the number four is written as IV. Because the one is before the five
we subtract it making four. The same principle applies to the number nine, which is written as
IX. There are six instances where subtraction is used:

- **I can be placed before V (5) and X (10) to make 4 and 9.**
- **X can be placed before L (50) and C (100) to make 40 and 90.**
- **C can be placed before D (500) and M (1000) to make 400 and 900.**

Given an integer, convert it to a roman numeral.

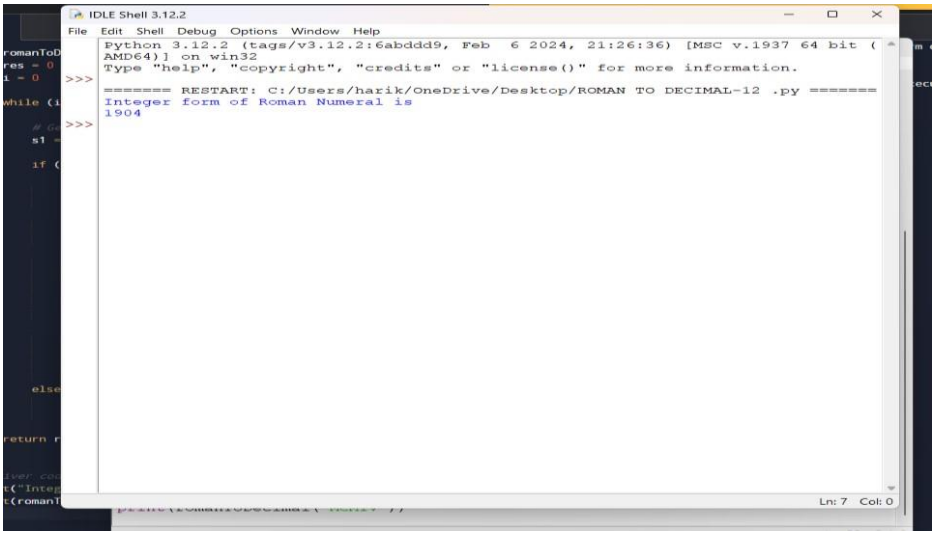
CODE:

```
def
value(r): if
```

```
(r == 'I'):
return 1 if
(r == 'V'):
return 5 if
(r == 'X'):
return 10
if (r ==
'L'): return
50 if (r ==
'C'): return
100 if (r
== 'D'):
return 500
if (r ==
'M'):
return
1000
return -1
def
romanToD
ecimal(str)
:
res = 0 i = 0 while (i <
len(str)): # Getting value
of symbol s[i] s1 =
value(str[i]) if (i + 1 <
len(str)):
# Getting value of symbol s[i +
1] s2 = value(str[i + 1]) #
```

```
Comparing both values if (s1
>= s2):
# Value of current symbol is
greater # or equal to the next
symbol res = res + s1 i = i + 1
else:
# Value of current symbol is greater #
or equal to the next symbol res = res +
s2 - s1 i = i + 2 else: res = res + s1 i = i +
1 return res # Driver code
print("Integer form of Roman Numeral
is"),
print(romanToDecimal("MCMIV"))
```

OUTPUT:



13. Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value

I 1

V 5
X 10
L 50
C 100
D 500
M 1000
For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as
XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.
Roman numerals are usually written largest to smallest from left to right. However, the numeral
for four is not IIII. Instead, the number four is written as IV. Because the one is before the five
we subtract it making four. The same principle applies to the number nine, which is written as
IX. There are six instances where subtraction is used:

- **I can be placed before V (5) and X (10) to make 4 and 9.**
- **X can be placed before L (50) and C (100) to make 40 and 90.**
- **C can be placed before D (500) and M (1000) to make 400 and 900.**

Code:

```
roman =
{'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
class Solution: def romanToInt(self, S: str) -> int:
summ= 0
for i in range(len(S)-1,-1,-1):
num = roman[S[i]]
if 3*num < summ:
summ = summ-
num else:
summ = summ+num
return sum
```

OUTPUT:

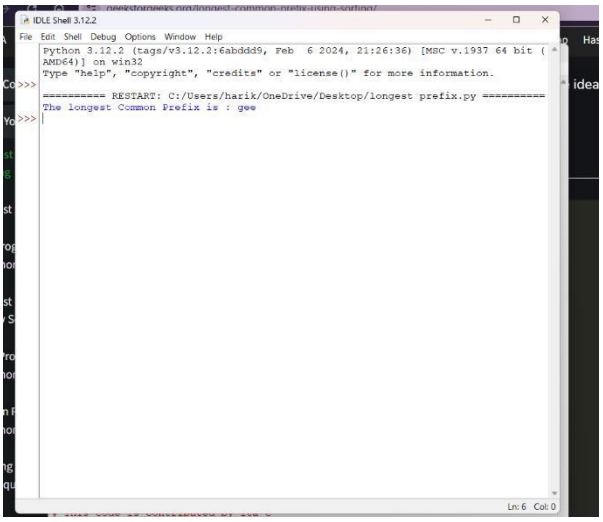
14. Longest Common Prefix
Write a function to find the longest common prefix string amongst an array of strings.
If there is no common prefix, return an empty string ""

CODE:

```
def longestCommonPrefix( a):

size = len(a)

# if size is 0, return empty
string if (size == 0):
return ""

if (size == 1):
return a[0]

# sort the array of strings
a.sort()

# find the minimum length from
# first and last string end =
min(len(a[0]), len(a[size - 1]))

# find the common prefix between
# the first and last
string i = 0 while (i <
end and a[0][i] ==
a[size - 1][i]):
```

```
i += 1

pre = a[0][0: i]
return pre

# Driver Code if
__name__ ==
"__main__":

input = ["geeksforgeeks", "geeks",
"geek", "geezer"]
print("The longest Common Prefix is :" ,
longestCommonPrefix(inp)
```

OUTPUT:



15. 3Sum

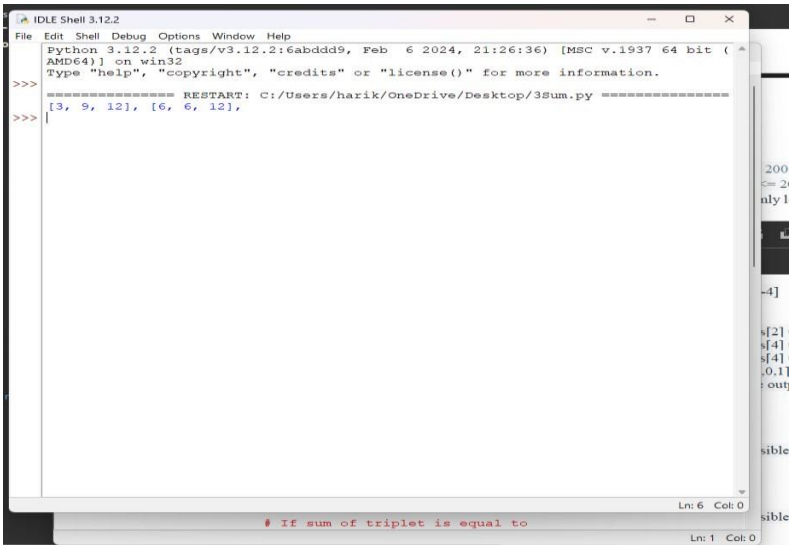
Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.
Notice that the solution set must not contain duplicate triplets.
Example 1:
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1)

= 0. nums[0] + nums[3] + nums[4] = (-1) + 2 +
(-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
CODE:

```
 def findTriplets(nums, n,
Sum):
i = 0
j = 0
k = 0
# list to store all unique triplets.
triplet = []
# list to store already found
triplets # to avoid duplication.
uniqTriplets = [] # Variable
used to hold triplet # converted
to string form. temp = "" #
Variable used to store current #
triplet which is stored in vector
# if it is unique. newTriplet =
[0, 0, 0] # Sort the input array.
nums.sort() # Iterate over the
array from the
# start and consider it as
the # first element. for i in
range(n - 2): # index of
the first element in # the
remaining elements. j = i
+ 1 # index of the last
element. k = n - 1
```

```
while(j < k):

# If sum of triplet is equal to
# given value, then check if
# this triplet is unique or not.
# To check uniqueness, convert
# triplet to string form and
# then check if this string is
# present in set or not. If # triplet is
unique, then store # it in list.
if(nums[i] + nums[j] + nums[k] ==
Sum):
temp = str(nums[i]) + ":" + str(nums[j]) + ":" +
str(nums[k]) if temp not in uniqTriplets:
uniqTriplets.append(temp) newTriplet[0] = nums[i]
newTriplet[1] = nums[j] newTriplet[2] = nums[k]
triplet.append(newTriplet) newTriplet = [0, 0, 0]

# Increment the first index
# and decrement the last #
index of remaining
elements.
j += 1
k -= 1

# If sum is greater than given
# value then to reduce sum #
decrement the last index. elif(nums[i]
+ nums[j] + nums[k] > Sum): k -= 1
```

```
# If sum is less than given value
# then to increase sum increment
# the first index of
remaining # elements.
else: j += 1
# If no unique triplet is found,
then # return 0. if(len(triplet)
== 0):
return 0

# Print all unique triplets
stored in # list. for i in
range(len(triplet)):
print(triplet[i], end = ", ")
return 1

# Driver Code nums =
[12, 3, 6, 1, 6, 9] n =
len(nums)
Sum = 24

# Function call if(not
findTriplets(nums, n, Sum)):
print("No triplets can be
formed.") output:
```



16. 3Sum Closest

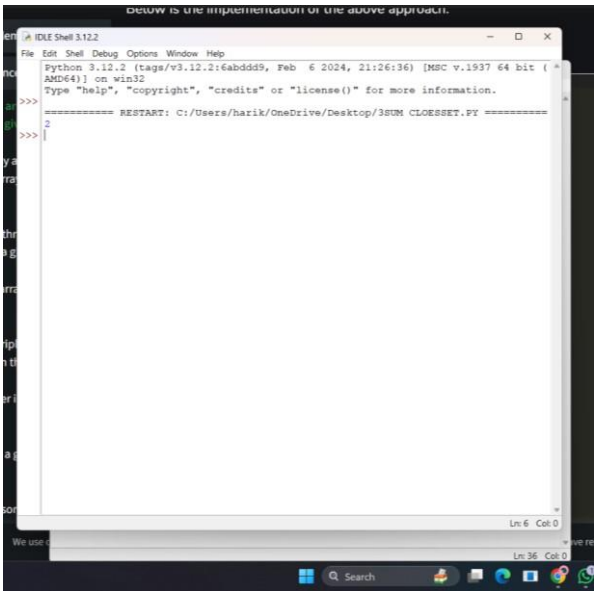
Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.
Return *the sum of the three integers*.
You may assume that each input would have exactly one solution.

CODE:

```
import sys


# Function to return the sum

of a # triplet which is closest

to x def solution(arr, x):


# To store the closest sum

closestSum = sys.maxsize


# Run three nested loops each

loop # for each element of

triplet for i in range (len(arr)) :

for j in range(i + 1, len(arr)):

for k in range(j + 1, len( arr)):
```

```
# Update the closestSum
if(abs(x - closestSum)
> abs(x - (arr[i] +
arr[j] + arr[k]))):
closestSum = (arr[i]
+ arr[j] + arr[k])

# Return the closest sum found
return closestSum

# Driver code if
__name__ ==
"__main__":

arr = [ -1, 2, 1, -4 ]
x = 1

print(solution(arr, x))
```

output:



17. Letter Combinations of a Phone Number
Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.
A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

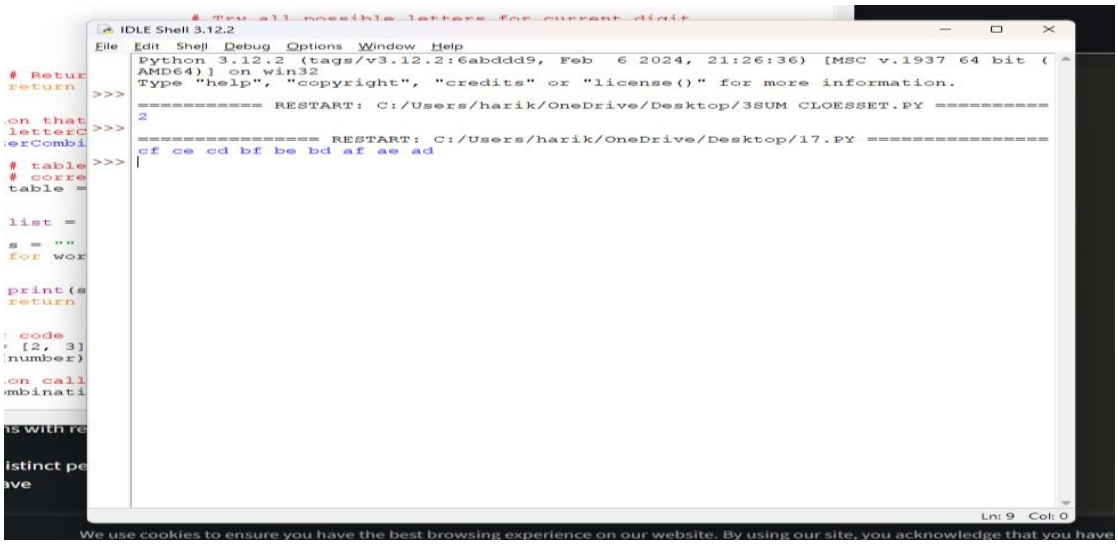
CODE:

```
# Python3 implementation of the approach

from collections import deque


# Function to return a list that contains

# all the generated letter combinations



def letterCombinationsUtil(number, n, table):


        list = []

q = deque()

        q.append("")


        while len(q) != 0:

                s = q.pop()


                # If complete word is

generated               # push it in the list

if len(s) == n:

list.append(s)                  else:


                        # Try all possible letters for current digit

                        # in number[]
```

```
                    for letter in table[number[len(s)]]:
                        q.append(s + letter)

        # Return the generated list
    return list


# Function that creates the
mapping and # calls
letterCombinationsUtil def
letterCombinations(number, n):

        # table[i] stores all characters that
    # corresponds to ith digit in phone
    table = ["0", "1", "abc", "def", "ghi", "jkl",
                        "mno", "pqrs", "tuv", "wxyz"]

    list = letterCombinationsUtil(number, n, table)

        s = ""  for
    word in list:
                s += word + " "

        print(s)
    return


# Driver code
number = [2, 3]
```

```
n =
len(number)


# Function call
letterCombinations(number, n)
```

OUTPUT:



18. 4Sum

Given an array nums of n integers, return *an array of all the unique quadruplets* [nums[a], nums[b], nums[c], nums[d]] such that:

- **0 <= a, b, c, d < n**
- **a, b, c, and d are distinct.**
- **nums[a] + nums[b] + nums[c] + nums[d] ==**

target CODE:

```
# Store the pair of
indices class Pair:
def __init__(self, x, y):
```

```
self.index1 = x
self.index2 = y

# Function to find the all the unique
quadruplets # with the elements at
different indices def
GetQuadruplets(nums, target):
# Store the sum mapped to a list of pair
indices map = {}

# Generate all possible pairs for the
map for i in range(len(nums) - 1):
for j in range(i + 1,
len(nums)): # Find the sum
of pairs of elements sum =
nums[i] + nums[j]

# If the sum doesn't exist then update with the
new pairs if sum not in map: map[sum] =
[Pair(i, j)]
# Otherwise, add the new pair of indices to the
current sum else:
map[sum].append(Pair(i, j))

# Store all the
Quadruplets ans = set()

for i in range(len(nums) - 1):
for j in range(i + 1,
```

```
len(nums)): lookUp = target -
(nums[i] + nums[j])

# If the sum with value (K - sum)
exists if lookUp in map:
# Get the pair of indices of
sum temp = map[lookUp]

for pair in temp:
# Check if i, j, k and l are distinct or not if pair.index1 != i and
pair.index1 != j and pair.index2 != i and pair.index2 != j:
l1 = [nums[pair.index1], nums[pair.index2], nums[i], nums[j]]

# Sort the list to avoid duplicacy
l1.sort()

# Update the set
ans.add(tuple(l1))

# Print all the Quadruplets
print(*reversed(list(ans)), sep = '\n')

# Driver Code
arr = [1, 0, -1, 0, -2, 2]
K = 0
GetQuadruplets(arr, K)
```

OUTPUT:

19. Remove Nth Node From End of List

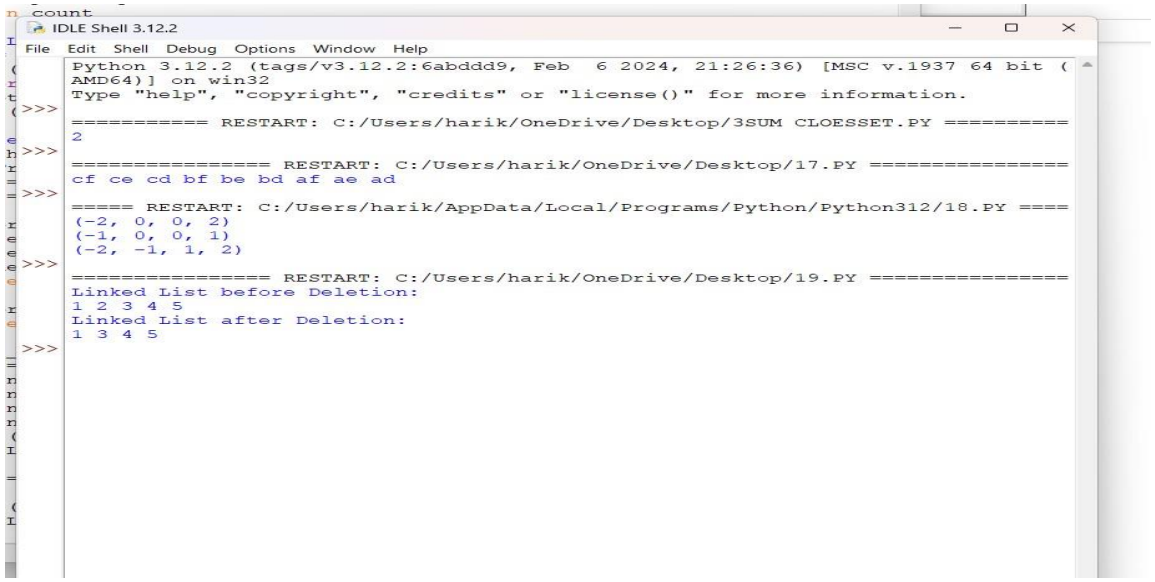
Given the head of a linked list, remove the nth node from the end of the list and return its head.

CODE:

```
# Python code for the deleting a node from end

# in two traversal


class Node:     def

__init__(self, value):

      self.data = value

self.next = None def

length(head): temp = head

count = 0 while(temp !=

None): count += 1 temp =

temp.next return count


def printList(head):

ptr = head while(ptr

!= None): print

(ptr.data, end =" ")

ptr = ptr.next print()
```

```
def deleteNthNodeFromEnd(head,
n): Length = length(head)
nodeFromBeginning = Length - n
+ 1 prev = None temp = head for i
in range(1, nodeFromBeginning):
prev = temp
temp =
temp.next
if(prev ==
None): head =
head.next
return head
else:
prev.next = prev.next.next
return head

if __name__ == '__main__':
head = Node(1) head.next =
Node(2) head.next.next =
Node(3) head.next.next.next =
Node(4)
head.next.next.next.next =
Node(5) print("Linked List
before Deletion:")
printList(head)

head = deleteNthNodeFromEnd(head, 4)

print("Linked List after Deletion:")
printList(head)
```

OUTPUT:



20. Valid Parentheses
Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:

1. **Open brackets must be closed by the same type of brackets.**
2. **Open brackets must be closed in the correct order.**
3. **Every close bracket has a corresponding open bracket of the same type.**

CODE:

```
def
areBracketsBalanced(expr):
stack = []
# Traversing the
Expression for char in
expr: if char in ["(", "{",
"["]:
```

```
# Push the element in the
stack stack.append(char)
else:
# IF current character is not
opening # bracket, then it must be
closing. # So stack cannot be empty
at this point. if not stack: return
False current_char = stack.pop() if
current_char == '(':
if char != ")":
return False if
current_char ==
'{':
if char != "}":
return False if
current_char ==
'[':
if char != "]":
return False #
Check Empty Stack
if stack: return
False return True #
Driver Code
if __name__ == "__main__":
expr = "{()}[]"

# Function call if
areBracketsBalanced(expr)
:
```

```
print("Balanced")
else:
print("Not Balanced")
```

OUTPUT: