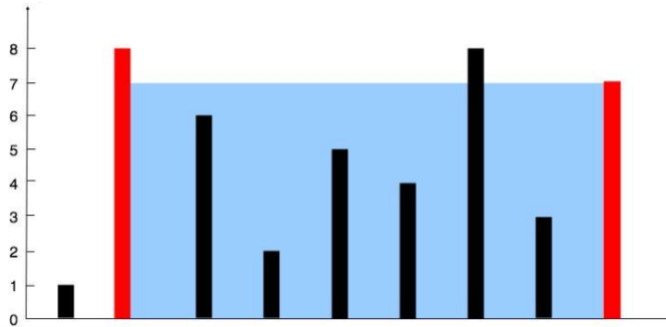


ASSIGNMENT-2

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.



Example 2:

Input: height = [1,8,6,2,5,4,8,3,7] Output: 49 Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

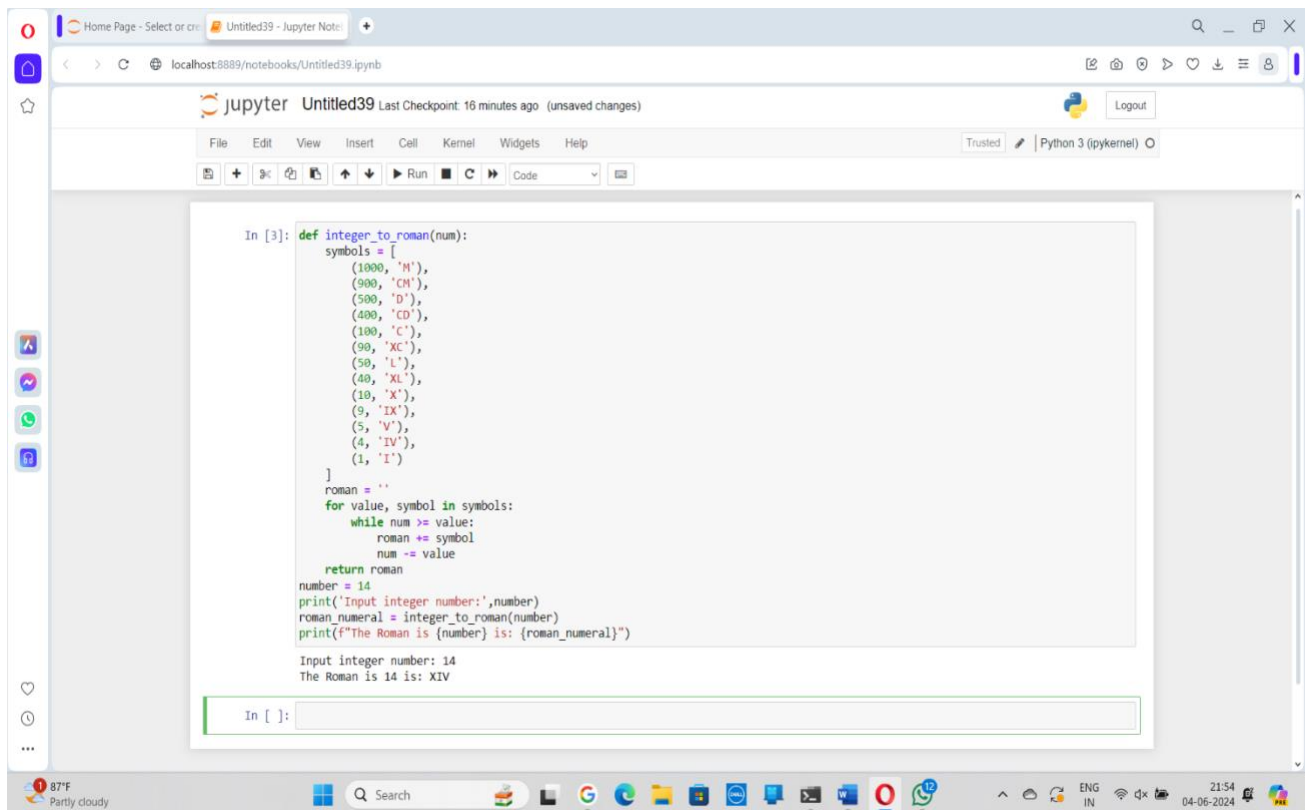
```
In [2]: def max_area(height):
        left, right = 0, len(height) - 1
        max_water = 0
        while left < right:
            width = right - left
            min_height = min(height[left], height[right])
            area = width * min_height
            max_water = max(max_water, area)
            if height[left] < height[right]:
                left += 1
            else:
                right -= 1
        return max_water
height = [1, 8, 6, 2, 5, 4, 8, 3, 7]
max_water_area = max_area(height)
print(max_water_area)

49
```

12. Integer to Roman 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 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.

Example 1: Input: num = 3 Output: "III" Explanation: 3 is represented as 3 ones.



```

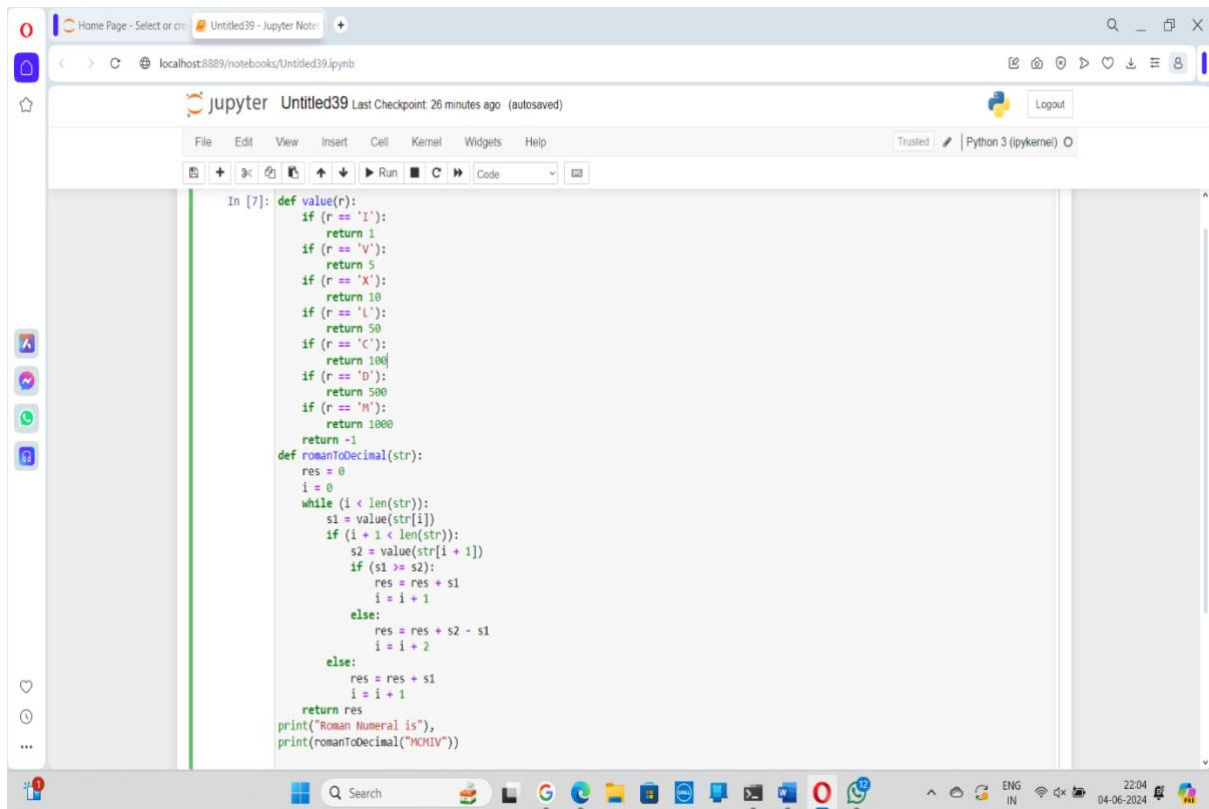
In [3]: def integer_to_roman(num):
        symbols = [
            (1000, 'M'),
            (900, 'CM'),
            (500, 'D'),
            (400, 'CD'),
            (100, 'C'),
            (90, 'XC'),
            (50, 'L'),
            (40, 'XL'),
            (10, 'X'),
            (9, 'IX'),
            (5, 'V'),
            (4, 'IV'),
            (1, 'I')
        ]
        roman = ''
        for value, symbol in symbols:
            while num >= value:
                roman += symbol
                num -= value
        return roman
number = 14
print('Input integer number:', number)
roman_numeral = integer_to_roman(number)
print(f"The Roman is {number} is: {roman_numeral}")

Input integer number: 14
The Roman is 14 is: XIV

```

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. Given a roman numeral, convert it to an integer.

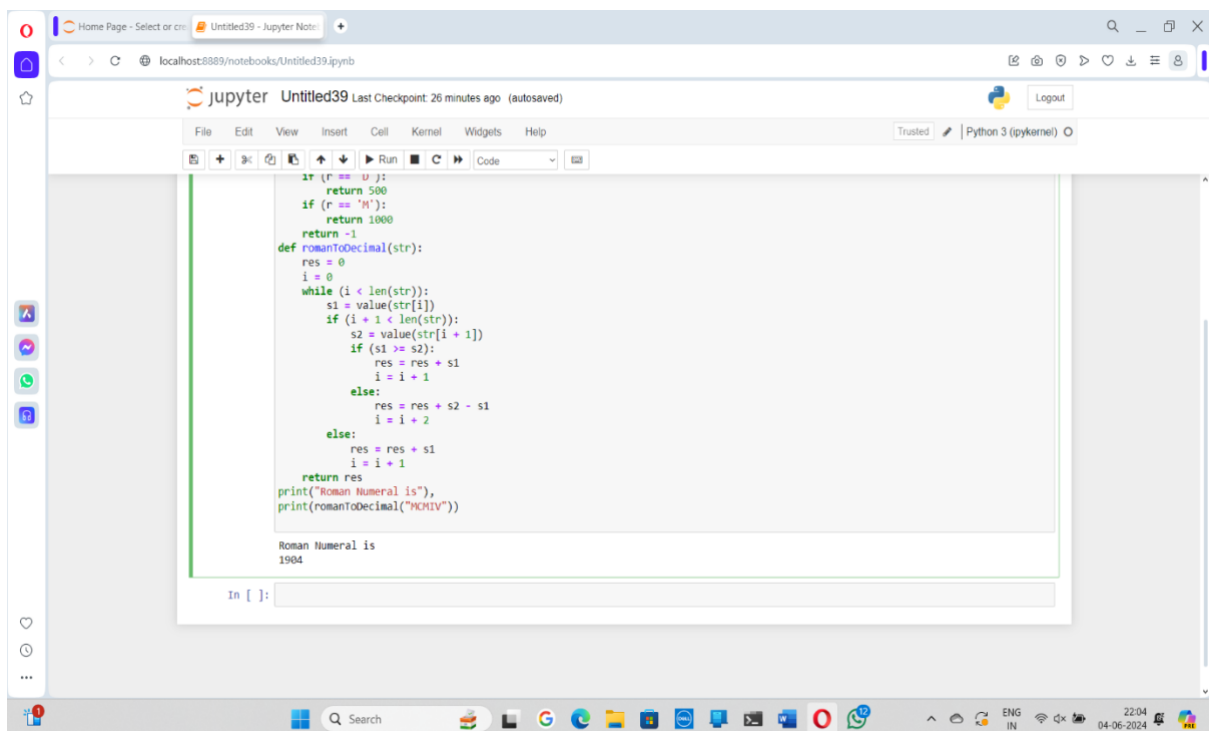
Example 1: Input: s = "III" Output: 3 Explanation: III = 3.



A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8889/notebooks/Untitled39.ipynb'. The notebook title is 'Untitled39' with a 'Last Checkpoint: 26 minutes ago (autosaved)' status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code editor contains the following Python code:

```
In [7]: 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 romanToDecimal(str):
        res = 0
        i = 0
        while (i < len(str)):
            s1 = value(str[i])
            if (i + 1 < len(str)):
                s2 = value(str[i + 1])
                if (s1 >= s2):
                    res = res + s1
                    i = i + 1
                else:
                    res = res + s2 - s1
                    i = i + 2
            else:
                res = res + s1
                i = i + 1
        return res
    print("Roman Numeral is"),
    print(romanToDecimal("MCMIV"))
```

The bottom of the window shows a Windows taskbar with a search bar, various application icons, and system tray information including 'ENG IN', '22:04', and '04-06-2024'.



A screenshot of the same Jupyter Notebook interface, showing the execution of the code. The code editor content is identical to the previous screenshot. Below the code editor, the output area displays the result of the execution:

```
Roman Numeral is
1904
```

The 'In []:' prompt is visible at the bottom of the output area. The rest of the interface, including the menu bar, toolbar, and Windows taskbar, remains the same as in the previous screenshot.

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 "".

Example 1: Input: strs = ["flower","flow","flight"] Output: "fl"

```
In [8]: def longestCommonPrefix(S):
        if "" in S or S == []:
            return ""
        preix = S[0]
        for i in range(1,len(S)):
            while(preix != ""):
                try:
                    if str.index(str(S[i]),preix) == 0:
                        break
                except:
                    preix = preix[:-1]
            preix = preix[:-1]
        return preix
    if __name__ == "__main__":
        s = ["flower","flight"]
        print(longestCommonPrefix(s))

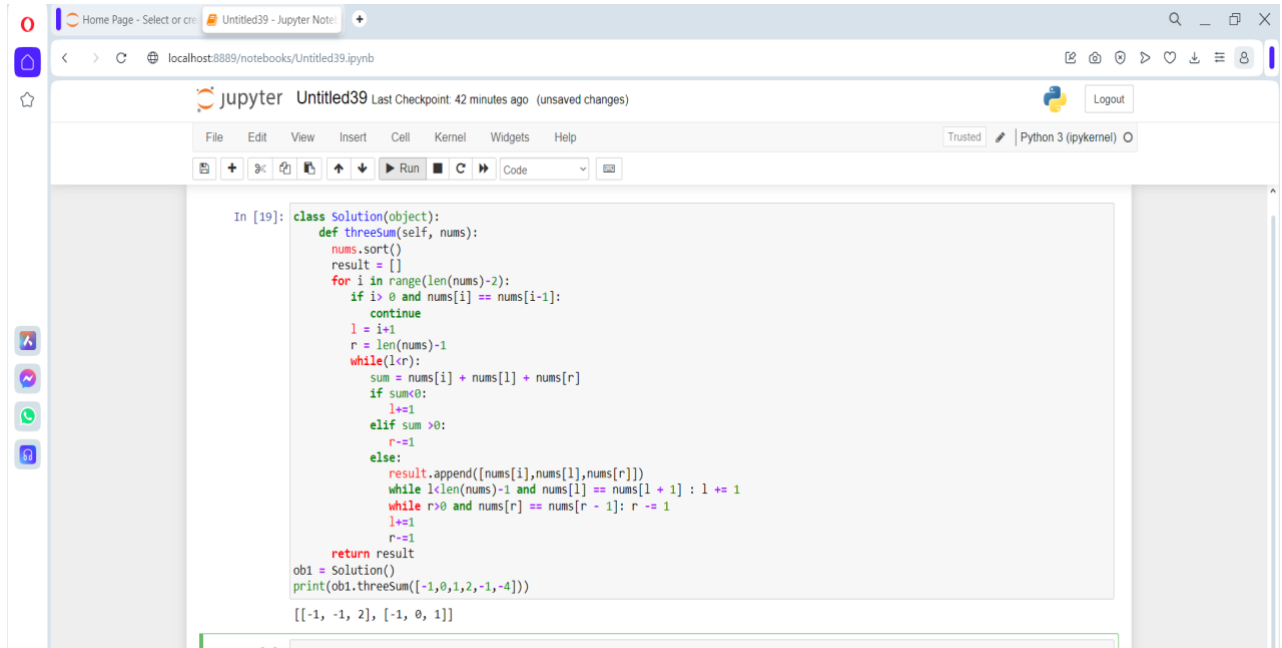
fl

In [ ]:
```

15. 3 Sum Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that $i \neq j$, $i \neq k$, and $j \neq k$, 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: $nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0$. $nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0$.

$\text{nums}[0] + \text{nums}[3] + \text{nums}[4] = (-1) + 2 + (-1) = 0$. 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



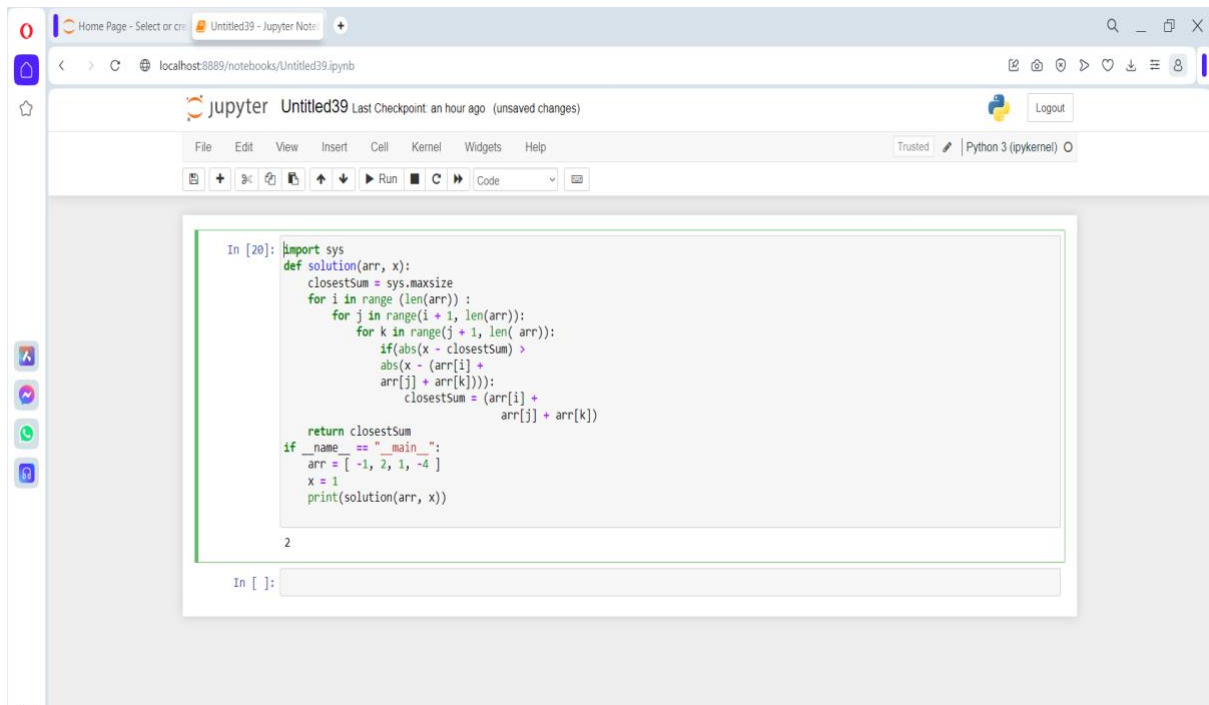
The screenshot shows a Jupyter Notebook titled 'Untitled39' running on a local host. The notebook contains a Python class 'Solution' with a method 'threeSum'. The method sorts the input array 'nums' and then iterates through it. For each element 'nums[i]', it uses two pointers 'l' and 'r' to find two other elements such that their sum with 'nums[i]' is zero. The found triplets are stored in a 'result' list. Finally, the method returns the result list. The output of the code is shown at the bottom: `[[[-1, -1, 2], [-1, 0, 1]]]`.

```
In [19]: class Solution(object):
def threeSum(self, nums):
    nums.sort()
    result = []
    for i in range(len(nums)-2):
        if i > 0 and nums[i] == nums[i-1]:
            continue
        l = i+1
        r = len(nums)-1
        while(l<r):
            sum = nums[i] + nums[l] + nums[r]
            if sum<0:
                l+=1
            elif sum >0:
                r-=1
            else:
                result.append([nums[i],nums[l],nums[r]])
                while l<len(nums)-1 and nums[l] == nums[l + 1] : l += 1
                while r>0 and nums[r] == nums[r - 1]: r -= 1
                l+=1
                r-=1
        return result
    ob1 = Solution()
    print(ob1.threeSum([-1,0,1,2,-1,-4]))

[[[-1, -1, 2], [-1, 0, 1]]]
```

16. 3 Sum 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.

Example 1: Input: `nums = [-1,2,1,-4]`, `target = 1` Output: 2 Explanation: The sum that is closest to the target is 2. $(-1 + 2 + 1 = 2)$.



```
In [20]: import sys
def solution(arr, x):
    closestSum = sys.maxsize
    for i in range(len(arr)):
        for j in range(i + 1, len(arr)):
            for k in range(j + 1, len(arr)):
                if(abs(x - closestSum) >
                    abs(x - (arr[i] +
                        arr[j] + arr[k]))):
                    closestSum = (arr[i] +
                        arr[j] + arr[k])
    return closestSum
if __name__ == "__main__":
    arr = [ -1, 2, 1, -4 ]
    x = 1
    print(solution(arr, x))

2

In [ ]:
```

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.



Example 1: Input: digits = "23" Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

```

In [25]: class Solution(object):
def letterCombinations(self, digits):
    if len(digits) == 0:
        return []
    characters = {2:"abc",3:"def",4:"ghi",5:"jkl",6:"mno",7:"pqrs",8:"tuv",9:"wxyz"}
    result = []
    self.solve(digits, characters, result)
    return result
def solve(self, digits, characters, result, current_string="", current_level = 0):
    if current_level == len(digits):
        result.append(current_string)
        return
    for i in characters[int(digits[current_level])]:
        self.solve(digits, characters, result, current_string+i, current_level+1)
ob1 = Solution()
print(ob1.letterCombinations("37"))

['dp', 'dq', 'dr', 'ds', 'ep', 'eq', 'er', 'es', 'fp', 'fq', 'fr', 'fs']

```

18.4 Sum 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:
 • $0 \leq a, b, c, d < n$
 • `a, b, c, and d` are distinct.
 • `nums[a] + nums[b] + nums[c] + nums[d] == target`
 You may return the answer in any order.

Example 1: Input: `nums = [1,0,-1,0,-2,2]`, `target = 0` Output: `[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]`

```

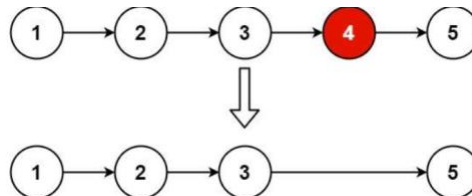
length = len(myArr)
size = ((length * (length - 1)) // 2)
aux = [None for _ in range(size)]
k = 0
for i in range(length - 1):
    for j in range(i + 1, length):
        aux[k] = pairSum()
        aux[k].sum = myArr[i] + myArr[j]
        aux[k].first = i
        aux[k].sec = j
        k += 1
aux.sort(key=lambda x: x.sum)
i = 0
j = size - 1
while (i < size and j >= 0):
    if ((aux[i].sum + aux[j].sum == sum)
        and noCommon(aux[i], aux[j])):
        print(myArr[aux[i].first], myArr[aux[i].sec],
              myArr[aux[j].first], myArr[aux[j].sec], sep=" ")
        return
    elif (aux[i].sum + aux[j].sum < sum):
        i += 1
    else:
        j -= 1
arr = [10, 20, 30, 40, 1, 2]
X = 9
findFourElements(arr, X)

20, 1, 30, 40

```

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.

Example 1: Input: head = [1,2,3,4,5], n = 2 Output: [1,2,3,5]



```

In [3]: 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)
```


The screenshot shows a Jupyter Notebook titled 'Untitled39' running on a local host. The code defines a linked list and a function to delete a node from the end. The output shows the linked list before and after deletion.

```
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 = deleteNodeFromEnd(head, 4)
    print("Linked List after Deletion:")
    printList(head)
```

Linked List before Deletion:
1 2 3 4 5
Linked List after Deletion:
1 3 4 5

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.

Example 1: Input: *s* = "()" Output: true

Home Page - Select or create a new notebook

Untitled39 - Jupyter Notebook

localhost:8889/notebooks/Untitled39.ipynb

Search

Home

Logout

File Edit View Insert Cell Kernel Widgets Help

Python 3 (ipykernel)

In [2]:

```
def isbalanced(s):
    while len(s) != 0:
        s = s.replace('()', '')
        s = s.replace('[]', '')
        s = s.replace('{}', '')
    if len(s) == 0:
        return True
    else:
        return False
s = input("Enter a string of brackets:")
print("Given string is balanced:", isbalanced(s))

Enter a string of brackets:{}
Given string is balanced: True
```

In []:

In []: