

DATE :

11/06/2024

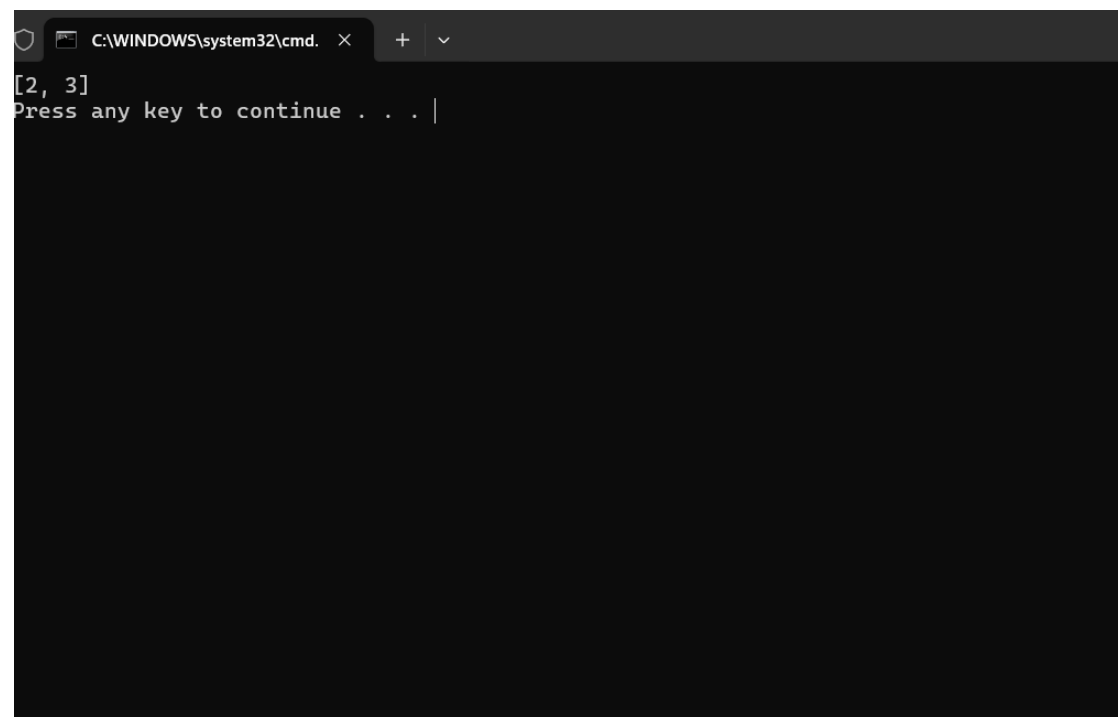
1) Two sum

CODE:

```
def two_sum(nums, target):  
    temp= {}  
    for i in range(len(nums)):  
        complement = target - nums[i]  
        if complement in temp:  
            return [temp[complement], i]  
        temp[nums[i]] = i  
    return None
```

```
nums = [2, 7, 11, 15]  
target = 26  
result = two_sum(nums, target)  
print(result)
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.' and standard window controls. The command prompt displays the output '[2, 3]' on the first line and 'Press any key to continue . . . |' on the second line, with a cursor at the end of the second line. The background is black and the text is white.

2) Add two numbers:

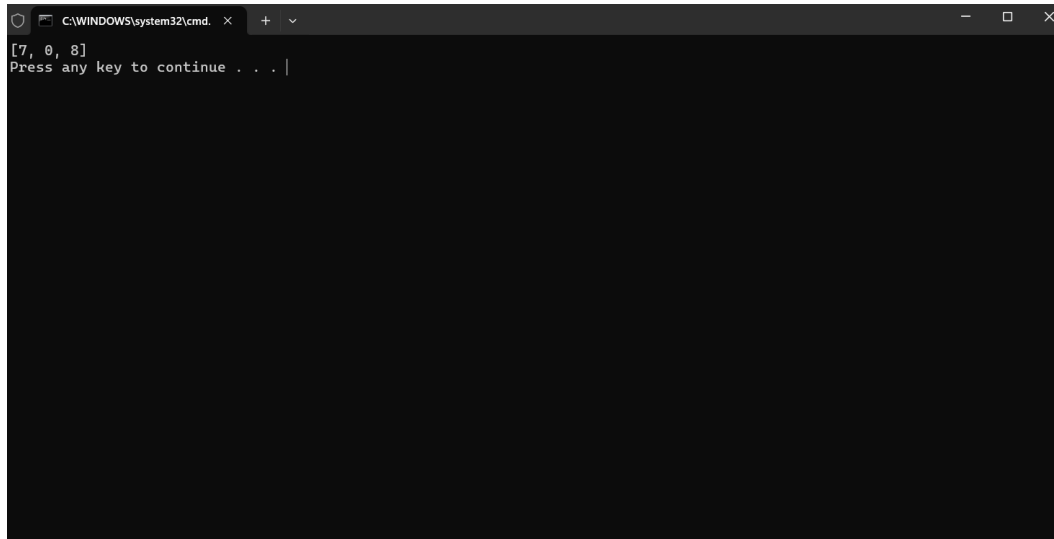
CODE:

```
def add(a,b):  
    a.reverse()  
    b.reverse()  
    anum=int("".join(map(str,a)))  
    bnum=int("".join(map(str,b)))  
    c=[]  
    d=anum+bnum  
    while d>0:
```

```

        r=d%10
        c.append(r)
        d=d//10
    return c
a=[2,4,3]
b=[5,6,4]
print(add(a,b))
OUTfLUT:

```



```

C:\WINDOWS\system32\cmd. x + v
[7, 0, 8]
Press any key to continue . . . |

```

3) Median of 2 sorted arrays:

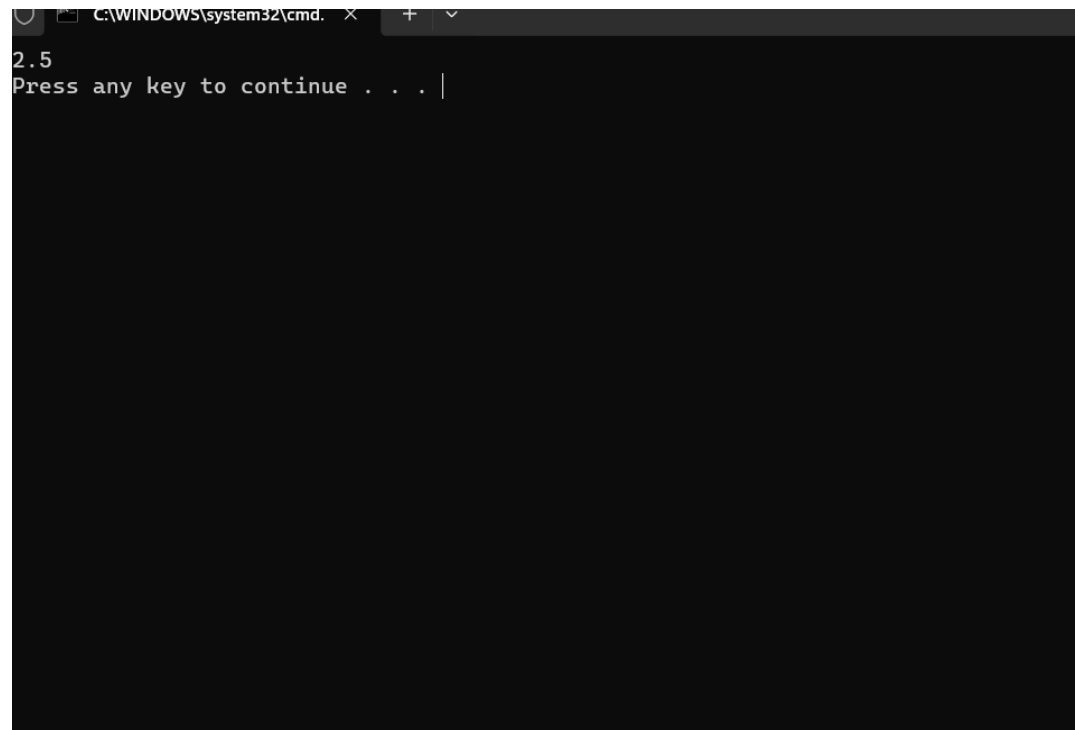
CODE:

```

def median(nums1, nums2):
    merged = sorted(nums1 + nums2)
    n = len(merged)
    if n % 2 == 0:
        return (merged[n // 2 - 1] + merged[n // 2]) / 2
    else:
        return merged[n // 2]
nums1 = [1, 2]
nums2 = [3,4]
print(median(nums1, nums2))

```

OUTfLUT:



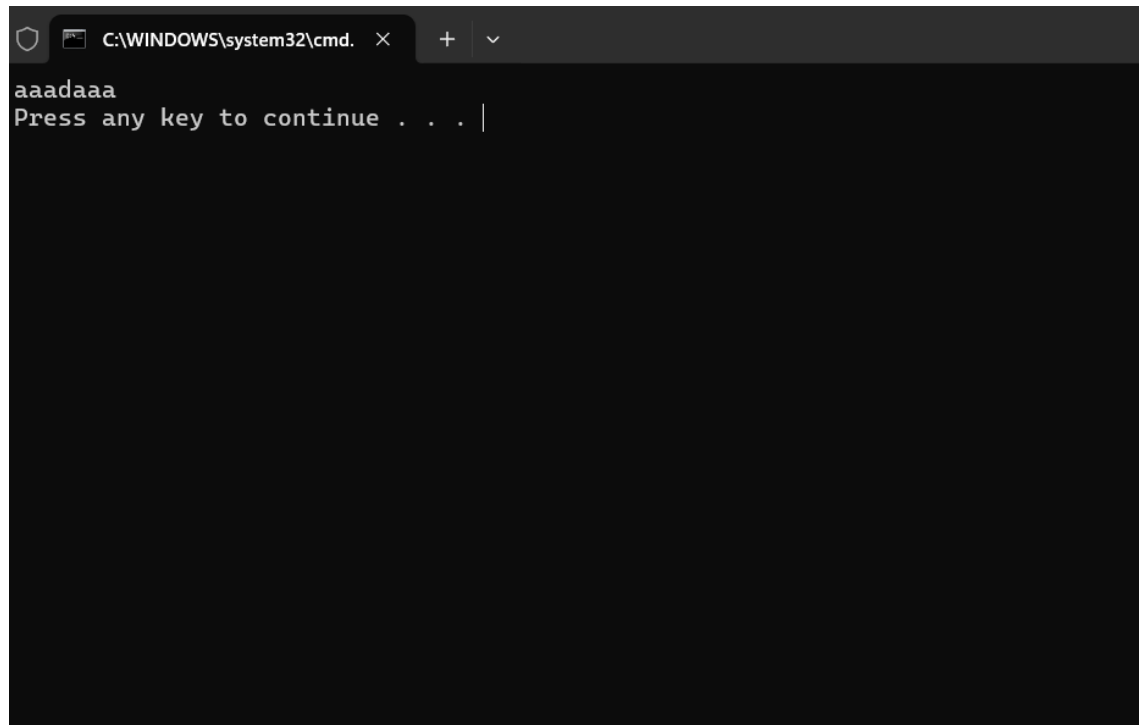
```
C:\WINDOWS\system32\cmd. X + v
2.5
Press any key to continue . . . |
```

4) Longest substring palindrome:

CODE:

```
def palin(s):
    maxpalin=""
    for i in range(len(s)):
        for j in range(i, len(s)):
            substr=s[i:j+1]
            if substr==substr[::-1] and len(substr)>len(maxpalin):
                maxpalin=substr
    return maxpalin
string="babaaadaaaa"
print(palin(string))
```

OUTfLUT:

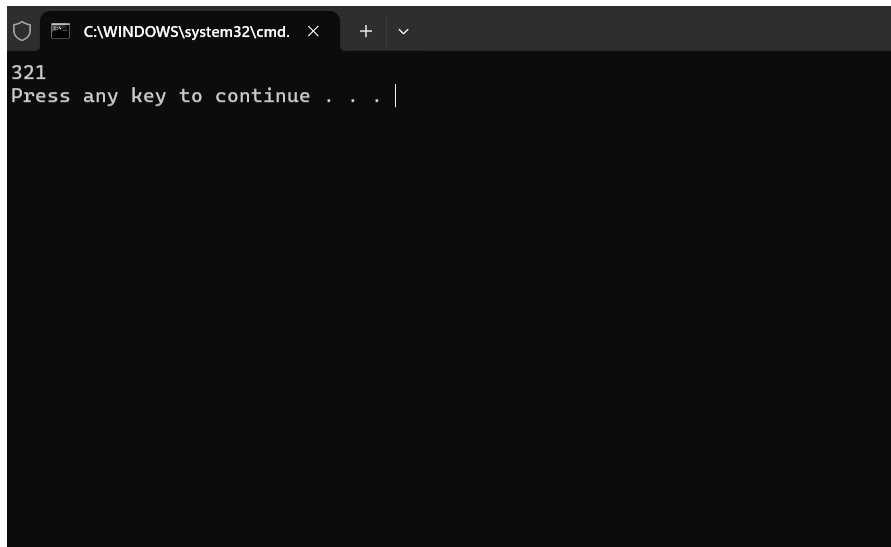
A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.' with a close button. The command prompt displays the text 'aaadaaa' on the first line and 'Press any key to continue . . . |' on the second line, with a vertical cursor at the end of the second line.

5) Reverse a number:

CODE:

```
def rev(num):  
    n=0  
    while num>0:  
        r=num%10  
        n=(n*10)+r  
        num=num//10  
    return n  
a=123  
print(rev(a))
```

OUTfLUT:



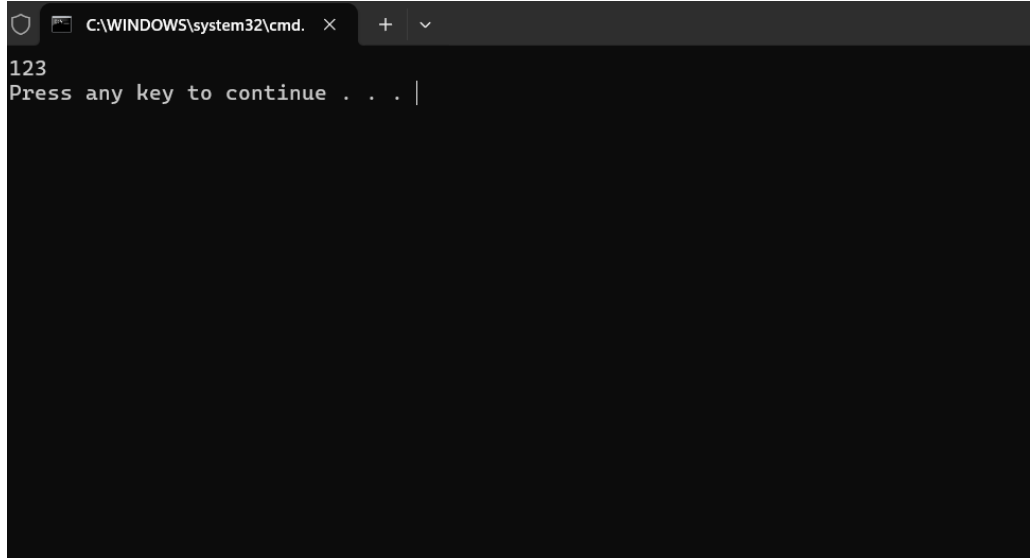
```
C:\WINDOWS\system32\cmd.
321
Press any key to continue . . . |
```

6) String to int:

CODE:

```
def string(str):
    return int(str)
a="123"
print(string(a))
```

OUTfLUT:



```
C:\WINDOWS\system32\cmd.
123
Press any key to continue . . . |
```

7) flalindrome or not number:

CODE:

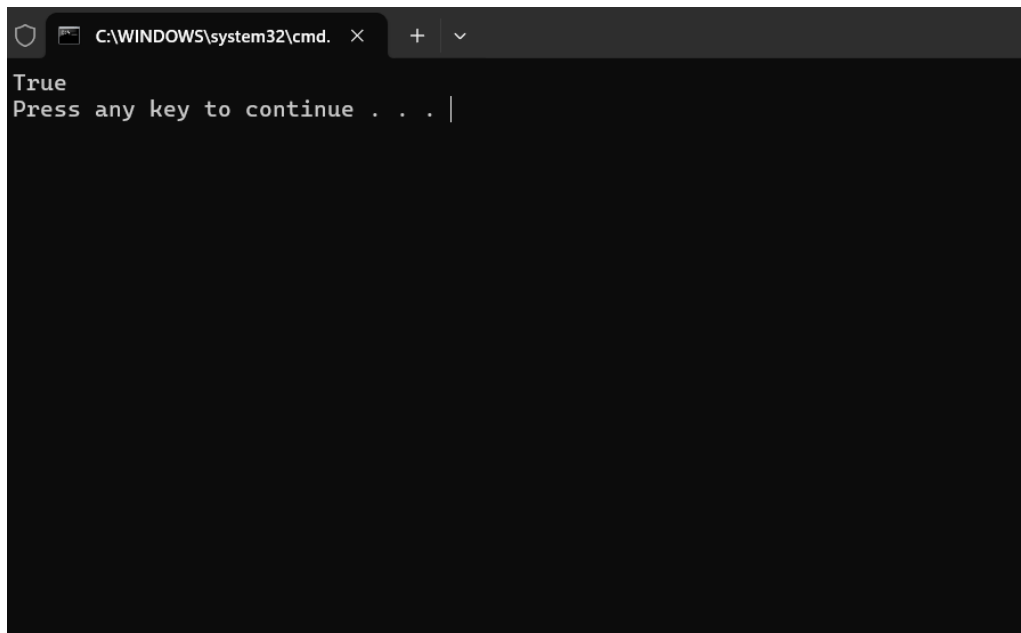
```
def rev(num):
    og=num
    n=0
    while num>0:
        r=num%10
```

```

        n=(n*10)+r
        num=num//10
    if n==og:
        return True
    else:
        return False
a=121
print(rev(a))

```

OUTPUT:



```

C:\WINDOWS\system32\cmd.
True
Press any key to continue . . . |

```

8) Longest substring without repeating chars:

CODE:

```

def length_of_longest_substring(s):
    char_index = {}
    start = 0
    max_length = 0

    for end in range(len(s)):
        if s[end] in char_index:
            start = max(start, char_index[s[end]] + 1)

        char_index[s[end]] = end
        max_length = max(max_length, end - start + 1)

    return max_length

s = "pwwkew"
print(length_of_longest_substring(s))

```

OUTPUT:

```
C:\WINDOWS\system32\cmd. x + v
3
Press any key to continue . . . |
```

9) Zigzag conversion:

CODE:

```
def convert(s, numRows):
    if numRows == 1 or numRows >= len(s):
        return s
```

```
    rows = [''] * numRows
    index, step = 0, 1
```

```
    for char in s:
        rows[index] += char
        if index == 0:
            step = 1
        elif index == numRows - 1:
            step = -1
        index += step
```

```
    return ''.join(rows)
```

```
a="fLAYfLALISHIRING"
```

```
b=4
```

```
print(convert(a,b))
```

OUTPUT:

```
C:\WINDOWS\system32\cmd. x + v
PINALSIGYHRPI
Press any key to continue . . . |
```

10) Regular Expression matching:

CODE:

```
import re
```

```
def is_match(s, p):
    pattern = re.compile(p)
    return bool(pattern.fullmatch(s))
```

```
s = "ab"
```

```
p = ".*"
```

```
print(is_match(s, p))
```

OUTPUT:

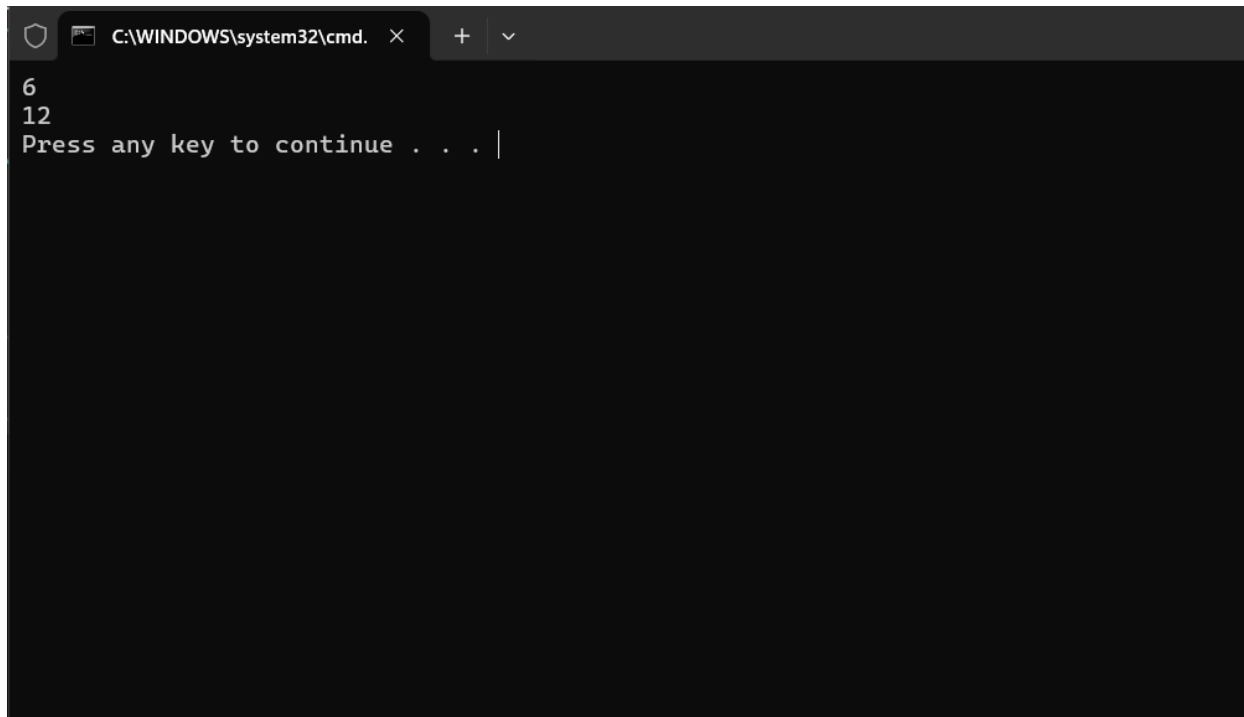

```
C:\WINDOWS\system32\cmd. x + v
True
Press any key to continue . . . |
```

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) :
        area = max(area, min(A[j], A[i]) * (j - i))
return area
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:



```
C:\WINDOWS\system32\cmd. x + v
6
12
Press any key to continue . . . |
```

12) 12. 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.

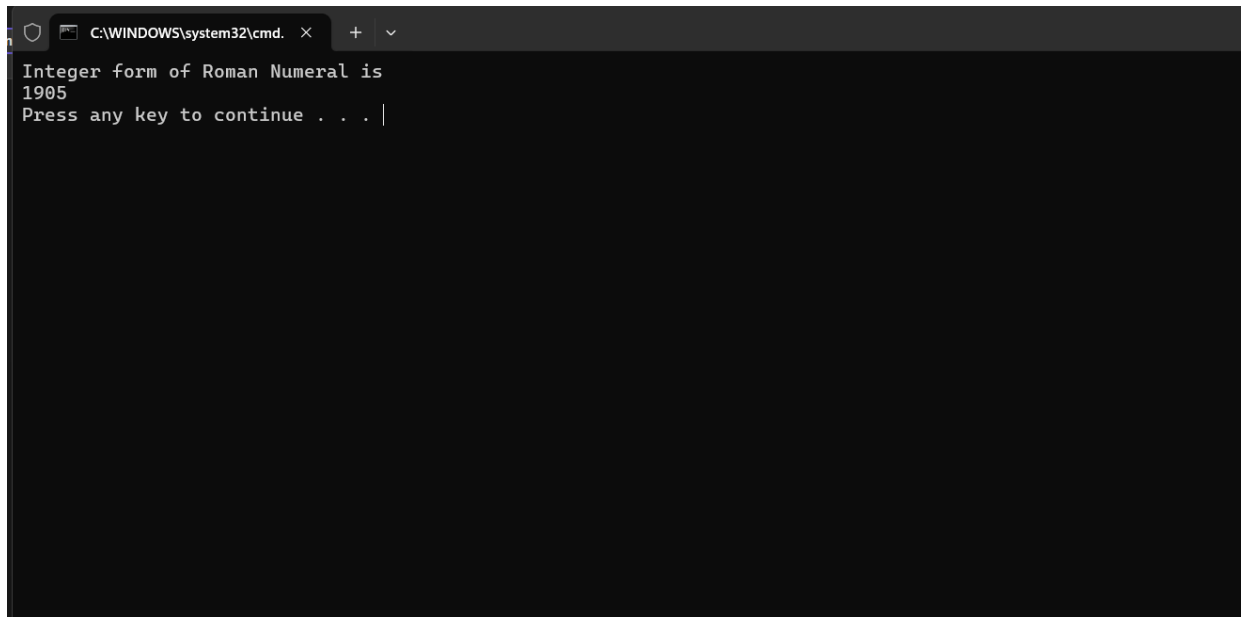
CODE:

```
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 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("Integer form of Roman Numeral is"),
print(romanToDecimal("MCMIV"))
```

OUTPUT:



```
C:\WINDOWS\system32\cmd. x + v
Integer form of Roman Numeral is
1905
Press any key to continue . . . |
```

13) 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
    prev_num = 0

    for i in range(len(S)-1, -1, -1):
        num = roman[S[i]]

        if num < prev_num:
            summ -= num
        else:
            summ += num

        prev_num = num

    return summ
```

OUTPUT:

```
C:\WINDOWS\system32\cmd. x + v
Integer form of Roman Numeral is
1905
Press any key to continue . . . |
```

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 == 0):
        return ""
    if (size == 1):
        return a[0]
a.sort()
end = min(len(a[0]), len(a[size - 1])) i =
0
    while (i < end and a[0][i] == a[size - 1][i]):
        i += 1
    pre = a[0][0: i]
return pre
if __name__ == "__main__":
    input = ["geeksforgeeks", "geeks",
"geek", "geezer"]
    print("The longest Common flrefix is : " ,
longestCommonPrefix(input))
```

OUTPUT:

```
C:\WINDOWS\system32\cmd. x + v
The longest Common Prefix is : gee
Press any key to continue . . . |
```

15) 3Sum 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: $nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0$. $nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0$. $nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0$. 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 triplet =
    []
    uniqTriplets = set()

    nums.sort()

    for i in range(n - 2):
        j = i + 1
        k = n - 1

        while j < k:
            if nums[i] + nums[j] + nums[k] == Sum:
                temp = str(nums[i]) + ":" + str(nums[j]) + ":" + str(nums[k])
                if temp not in uniqTriplets:
                    uniqTriplets.add(temp)
                    triplet.append([nums[i], nums[j], nums[k]])

                j += 1
                k -= 1
            elif nums[i] + nums[j] + nums[k] > Sum:
```

```

        k -= 1
    else:
        j += 1

    if not triplet:
        return 0

    for t in triplet:
        print(t, end=", ")

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

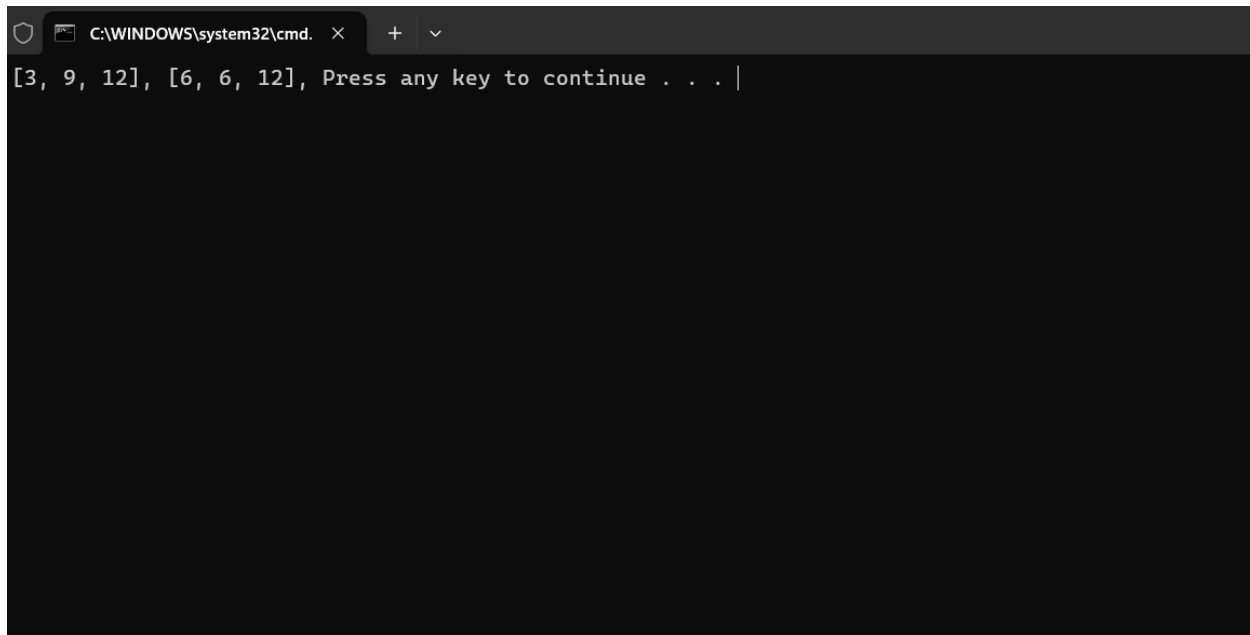
```

```

if not findTriplets(nums, n, Sum): print("No
triplets can be formed.")

```

OUTPUT:



```

C:\WINDOWS\system32\cmd. x + v
[3, 9, 12], [6, 6, 12], Press any key to continue . . . |

```

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`

```

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)):
            current_sum = arr[i] + arr[j] + arr[k]
            if abs(x - current_sum) < abs(x - closestSum):
                closestSum = current_sum

```

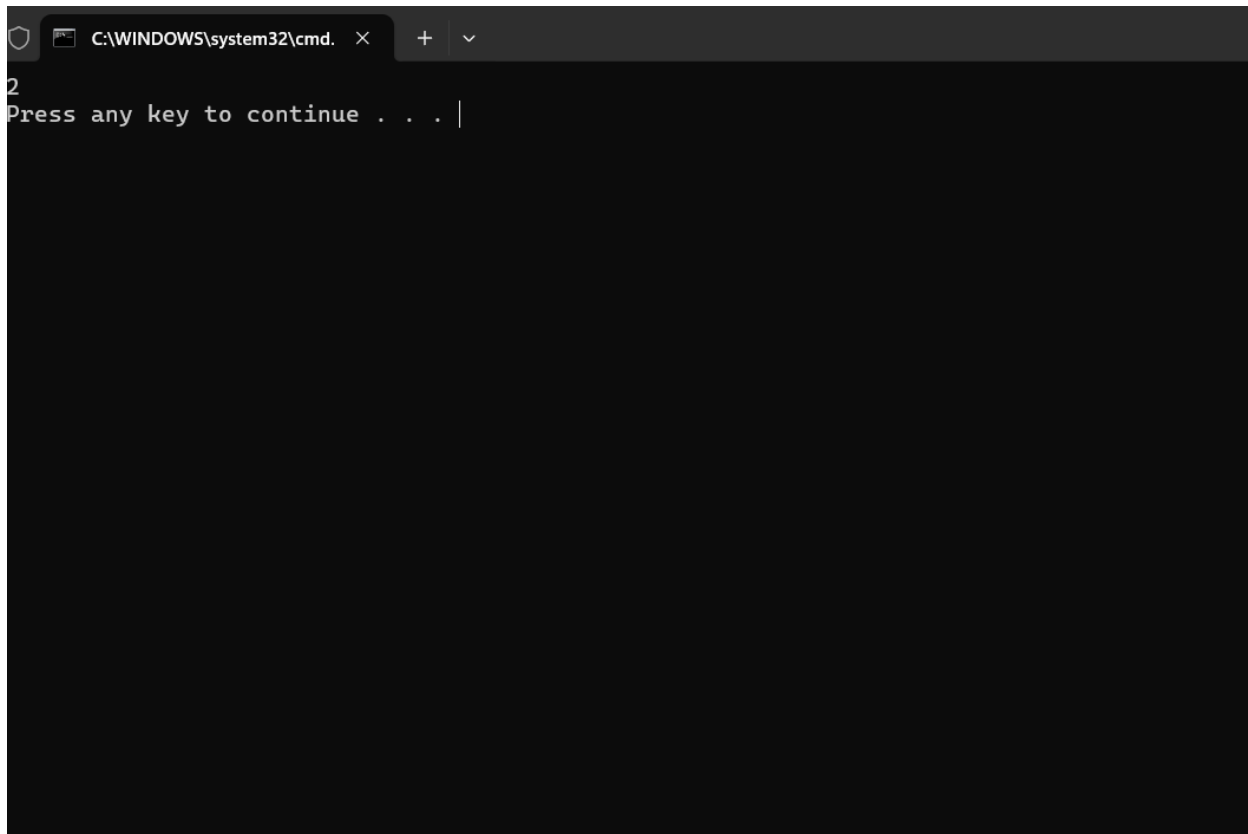
```

    return closestSum

if __name__ == "__main__":
    arr = [-1, 2, 1, -4]
    x = 1 print(solution(arr,
x))

```

OUTPUT:



```

C:\WINDOWS\system32\cmd.
2
Press any key to continue . . . |

```

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:

```

from collections import deque

def letterCombinationsUtil(number, n, table): result =
    []
    q = deque() q.append("")

    while q:
        s = q.popleft()

        if len(s) == n:

```



```

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

return result

def letterCombinations(number, n):
    table = ["0", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"]
    result = letterCombinationsUtil(number, n, table)

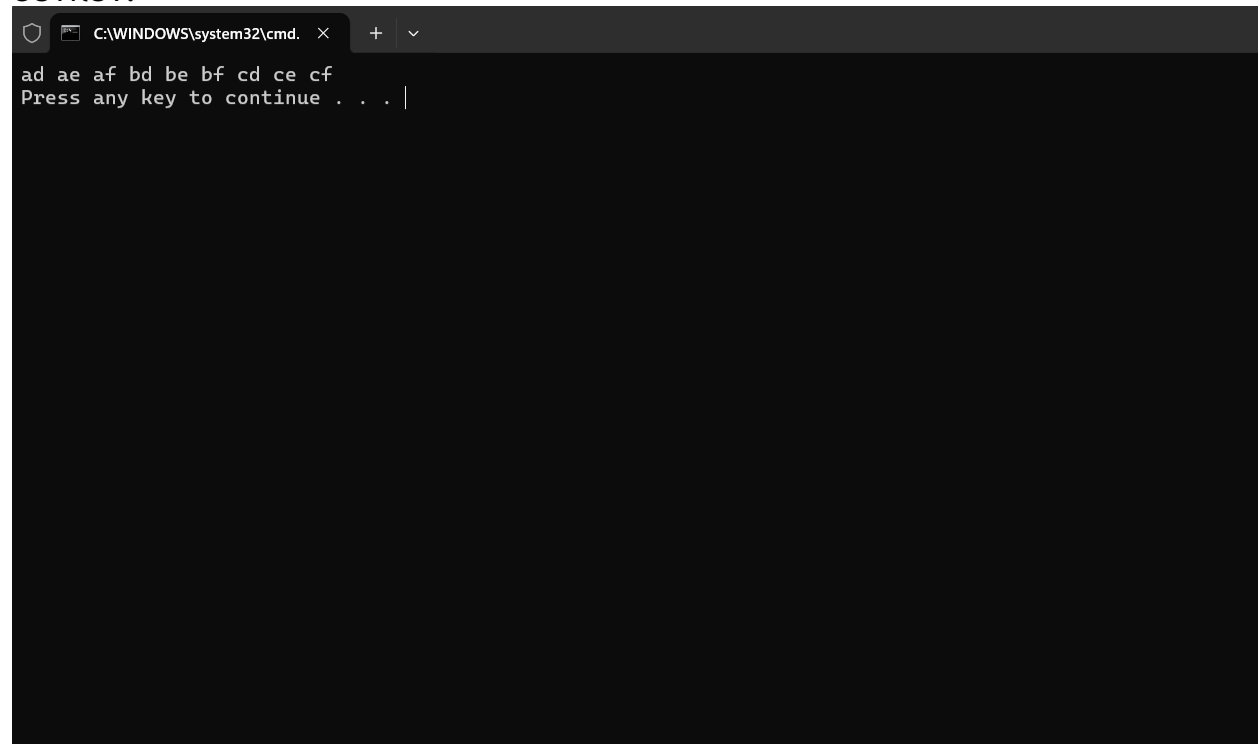
    output = ""
    for word in result:
        output += word + " "

    print(output)

number = ['2', '3']
n = len(number)

```

letterCombinations(number, n)
OUTPUT:



The screenshot shows a Windows command prompt window with the title bar 'C:\WINDOWS\system32\cmd.' and a tab icon. The command prompt displays the output of the program: 'ad ae af bd be bf cd ce cf' followed by a prompt 'Press any key to continue . . . |'.

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: • $0 \leq a, b, c, d < n$ • a, b, c, and d are distinct. • $\text{nums}[a] + \text{nums}[b] + \text{nums}[c] + \text{nums}[d] == \text{target}$

CODE:

```

class flair:
    def __init__(self, x, y):

```

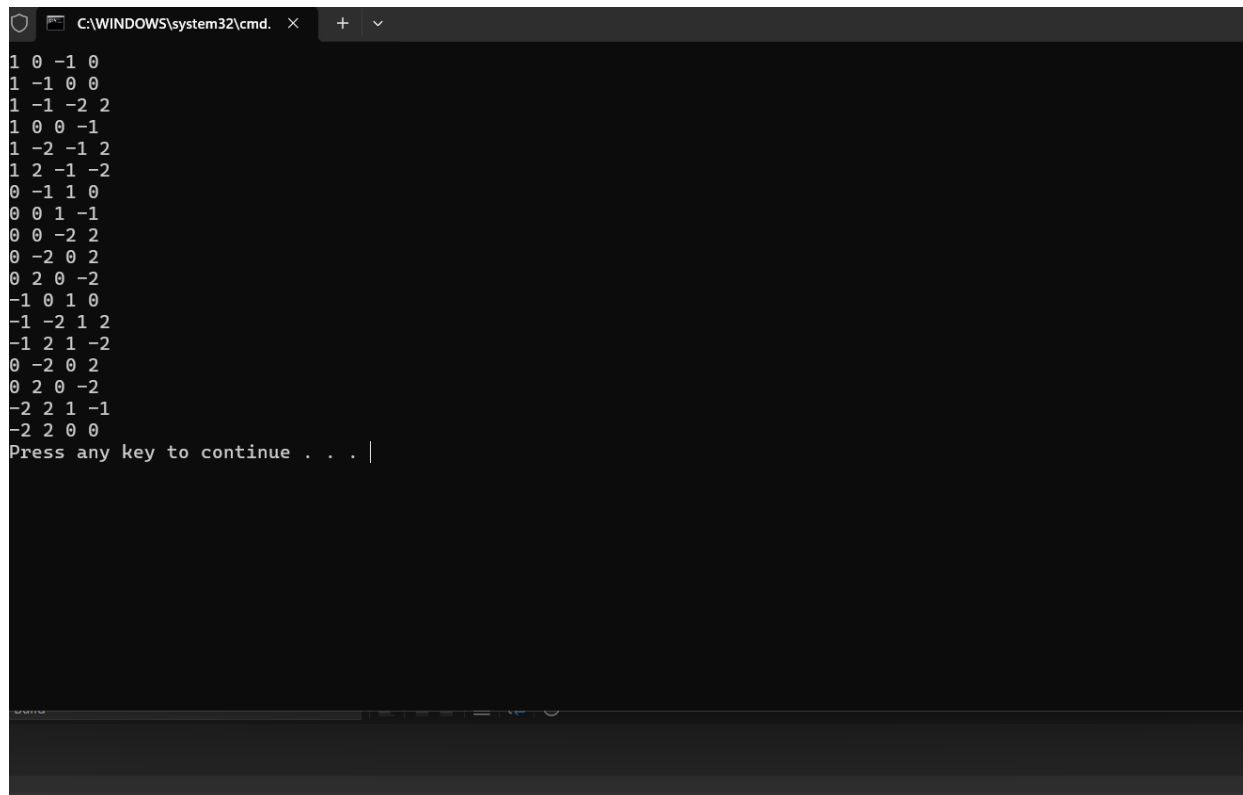
```

        self.index1 = x
        self.index2 = y

def GetQuadruplets(nums, target):
    map = {}
    ans = set()
    for i in range(len(nums) - 1):
        for j in range(i + 1, len(nums)):
            sum = nums[i] + nums[j]
            if sum not in map:
                map[sum] = [i, j]
            else:
                map[sum].append(i, j)
    for i in range(len(nums) - 1):
        for j in range(i + 1, len(nums)):
            lookUp = target - (nums[i] + nums[j])
            if lookUp in map:
                temp = map[lookUp]
                for pair in temp:
                    if pair.index1 != i and pair.index1 != j and pair.index2 != i
and pair.index2 != j:
                        print(nums[i], nums[j], nums[pair.index1],
nums[pair.index2])

arr = [1, 0, -1, 0, -2, 2]
K = 0
GetQuadruplets(arr, K)
OUTPUT:

```



```

C:\WINDOWS\system32\cmd. x + v
1 0 -1 0
1 -1 0 0
1 -1 -2 2
1 0 0 -1
1 -2 -1 2
1 2 -1 -2
0 -1 1 0
0 0 1 -1
0 0 -2 2
0 -2 0 2
0 2 0 -2
-1 0 1 0
-1 -2 1 2
-1 2 1 -2
0 -2 0 2
0 2 0 -2
-2 2 1 -1
-2 2 0 0
Press any key to continue . . . |

```

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:

```
class Node:
    def __init__(self, value):
        self.data = value
        self.next = None

def length(head): temp
= head count = 0
    while temp is not None:
        count += 1
        temp = temp.next
    return count

def printList(head): ptr =
head
    while ptr is not None:
        print(ptr.data, end=" ")
        ptr = ptr.next
    print()

def deleteNthNodeFromEnd(head, n): Length =
length(head) nodeFromBeginning = Length
- n + 1 prev = None
temp = head

    if nodeFromBeginning == 1:
        head = head.next
        return head

    for i in range(1, nodeFromBeginning):
        prev = temp
        temp = temp.next

    prev.next = temp.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:
```

```
C:\WINDOWS\system32\cmd. x + v
Linked List before Deletion:
1 2 3 4 5
Linked List after Deletion:
1 3 4 5
Press any key to continue . . . |
```

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.

OUTPUT:

```
def areBracketsBalanced(expr): stack =
    []
    opening_brackets = ["(", "{", "["]
    closing_brackets = [")", "}", "]"

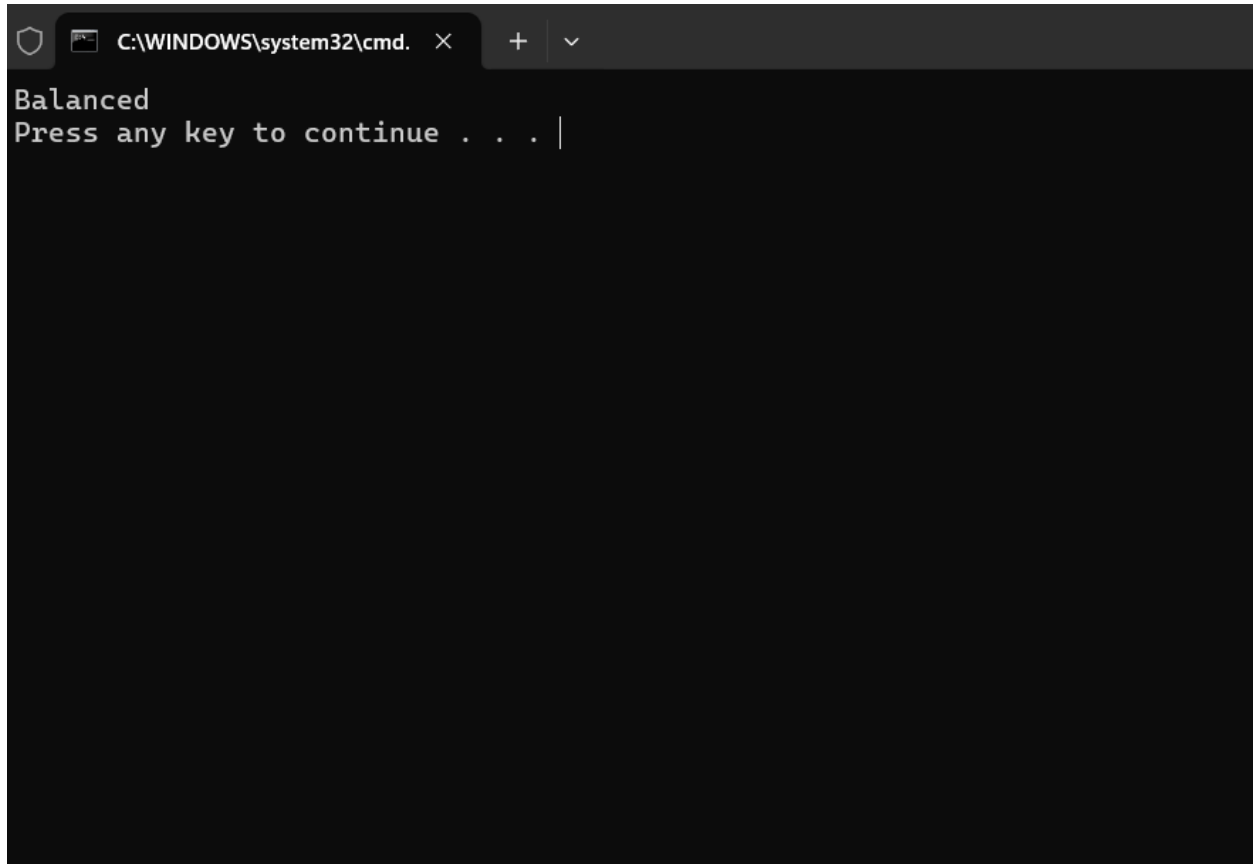
    for char in expr:
        if char in opening_brackets:
            stack.append(char)
        elif char in closing_brackets:
            if not stack:
                return False
            current_char = stack.pop()
            if opening_brackets.index(current_char) != closing_brackets.index(char):
                return False

    if stack:
        return False
    return True

if __name__ == "__main__":
    expr = "{(){}[]}"

    if areBracketsBalanced(expr):
```

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



A screenshot of a Windows command prompt window. The title bar shows the path "C:\WINDOWS\system32\cmd." and a close button. The window content displays the output of a program: "Balanced" on the first line, followed by "Press any key to continue . . . |" on the second line, where the vertical bar indicates the cursor is waiting for input.