**ASSIGNMENT -1**

1. Two Sum Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice.

PROGRAM :

import array as arr

ar=arr.array('i',[])

n=int(input("Enter The Array Size:-"))

target=int(input("Enter Your Target:-"))

print(f"Enter {n} Elements")

for i in range(n):

ar.insert(i,int(input()))

list1=[]

for i in range(len(ar)):

list1.append(i)

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

if ar[i]+ar[j]==target and len(list1)!=2:

list1.append(j)

if len(list1)==1:

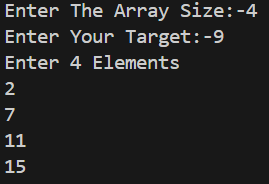
list1.clear()

else:

break

print(list1)

**INPUT**

****

**OUTPUT**



2. Add Two Numbers You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**PROGRAM**

class ListNode:

def \_init\_(self, val=0, next=None):

self.val = val

self.next = next

def addTwoNumbers(l1, l2):

dummy = ListNode()

current, carry = dummy, 0

while l1 or l2 or carry:

x = l1.val if l1 else 0

y = l2.val if l2 else 0

carry, out = divmod(x + y + carry, 10)

current.next = ListNode(out)

current = current.next

l1 = l1.next if l1 else None

l2 = l2.next if l2 else None

return dummy.next

def print\_linked\_list(node):

while node:

print(node.val, end=" -> " if node.next else "\n")

node = node.next

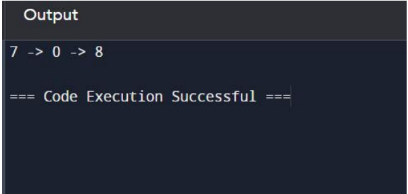
l1 = ListNode(2, ListNode(4, ListNode(3)))

l2 = ListNode(5, ListNode(6, ListNode(4)))

result = addTwoNumbers(l1, l2)

print\_linked\_list(result)

**OUTPUT**

****

3. Longest Substring without Repeating Characters Given a string s, find the length of the longest substring without repeating characters.

**PROGRAM**

def length\_of\_longest\_substring(s):

start = 0

max\_length = 0

char\_index\_map = {}

for end in range(len(s)):

if s[end] in char\_index\_map:

start = max(start, char\_index\_map[s[end]] + 1)

char\_index\_map[s[end]] = end

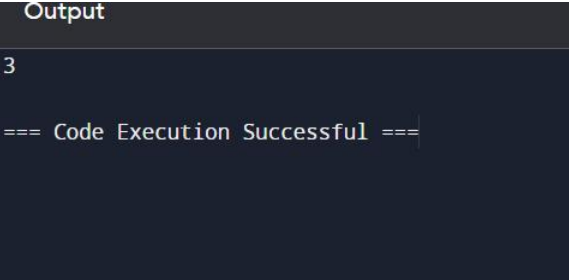
max\_length = max(max\_length, end - start + 1)

return max\_length

s = "abcabcbb"

print(length\_of\_longest\_substring(s))

**OUTPUT**



4. Median of Two Sorted Arrays Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

**PROGRAM**

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

nums1=self.nums1=nums1

nums2=self.nums2=nums2

nums3=nums1+nums2

nums3=sorted(nums3)

if len(nums3)%2==1:

median=len(nums3)//2

return nums3[median]

else:

ind=len(nums3)//2

median=(nums3[ind]+nums3[ind-1])/2

return median

nums1=[1,2]

nums2=[3,4]

sol=Solution()

result=sol.findMedianSortedArrays(nums1,nums2)

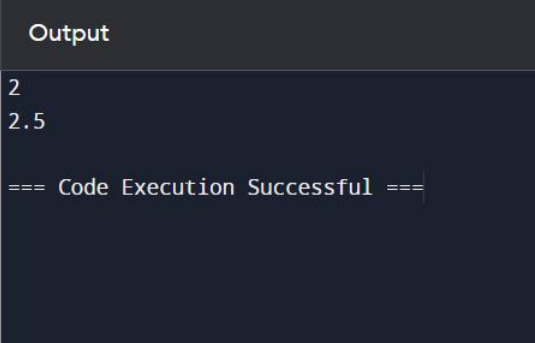
print(result)

**INPUT**

****

****

**OUTPUT**

****

5. Longest Palindromic Substring Given a string s, return the longest palindromic substring in s.

**PROGRAM**

def longest\_palindromic\_substring(s):

n = len(s)

dp = [[False] \* n for \_ in range(n)]

start = 0

max\_len = 1

for i in range(n):

dp[i][i] = True

for length in range(2, n + 1):

for i in range(n - length + 1):

j = i + length - 1

if s[i] == s[j] and (length == 2 or dp[i + 1][j - 1]):

dp[i][j] = True

if length > max\_len:

start = i

max\_len = length

return s[start:start + max\_len]

s = "babad"

print(longest\_palindromic\_substring(s))

**INPUT**

****

**OUTPUT**

****

6. Zigzag Conversion The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string and make this conversion given a number of rows: string convert(string s, int numRows).

**PROGRAM**

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)

input\_string = "PAYPALISHIRING"

num\_rows = 3

result = convert(input\_string, num\_rows)

print(result)

**INPUT**



**OUTPUT**

****

7. Reverse Integer Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

**PROGRAM**

def reverse(x: int) -> int:

if x < 0:

rev = int(str(x)[1:][::-1]) \* -1

else:

rev = int(str(x)[::-1])

if rev < -2\*\*31 or rev > 2\*\*31 - 1:

return 0

else:

return rev

num = 123

reversed\_num = reverse(num)

print(reversed\_num)

**INPUT**

****

**OUTPUT**

****

8. String to Integer (atoi) Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

**PROGRAM**

def myAtoi(s: str) -> int:

s = s.strip()

if not s:

return 0

sign = 1

if s[0] in ['-', '+']:

if s[0] == '-':

sign = -1

s = s[1:]

result = 0

for char in s:

if not char.isdigit():

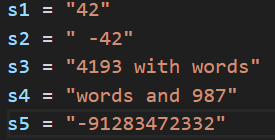
break

result = result \* 10 + int(char)

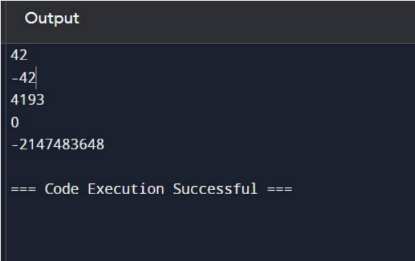
result = max(-2\*\*31, min(sign \* result, 2\*\*31 - 1))

return result

**INPUT**

****

**OUTPUT**

****

9. Palindrome Number Given an integer x, return true if x is a palindrome, and false otherwise.

**PROGRAM**

num=int(input("Enter The Number:-"))

num\_copy=num

str\_1=""

while num!=0:

rem=num%10

str\_2=str(rem)

str\_1+=str\_2

num//=10

num=int(str\_1)

if num==num\_copy:

print("True")

else:

print("False")

**IP/OP**

****

10. Regular Expression Matching Given an input string s and a pattern p, implement regular expression matching with support for '.' and '\*' where: ● '.' Matches any single character. ● '\*' Matches zero or more of the preceding element. The matching should cover the entire input string (not partial).

**PROGRAM**

def isMatch(s, p):

dp = [[False] \* (len(p) + 1) for \_ in range(len(s) + 1)]

dp[0][0] = True

for i in range(1, len(p) + 1):

if p[i - 1] == '\*':

dp[0][i] = dp[0][i - 2]

for i in range(1, len(s) + 1):

for j in range(1, len(p) + 1):

if p[j - 1] in {s[i - 1], '.'}:

dp[i][j] = dp[i - 1][j - 1]

elif p[j - 1] == '\*':

dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and p[j - 2] in {s[i - 1], '.'})

return dp[-1][-1]

s = "mississippi"

p = "mis\*is\*p\*."

print(isMatch(s, p))

**INPUT**

****

**OUTPUT**

****