**ASSIGNMENT-1**

**1.two sum**

def two\_sum(nums, target):

num\_dict = {}

for i, num in enumerate(nums):

complement = target - num

if complement in num\_dict:

return [num\_dict[complement], i]

num\_dict[num] = i

nums1 = [2, 7, 11, 15]

target1 = 9

print(two\_sum(nums1, target1))

**2.add two numbers**

num1 = 10

num2 = 20

sum = num1 + num2

print("The sum of", num1, "and", num2, "is:", sum)

**3.longest sumstring without repeating charctrers**

def lengthOfLongestSubstring(s):

start = maxLength = 0

usedChars = {}

for i, char in enumerate(s):

if char in usedChars and start <= usedChars[char]:

start = usedChars[char] + 1

else:

maxLength = max(maxLength, i - start + 1)

usedChars[char] = i

return maxLength

s = "abcabcbb"

print(lengthOfLongestSubstring(s))

**4.median of two sorted arrays**

from typing import List

class Solution:

def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:

nums = sorted(nums1 + nums2)

n = len(nums)

if n % 2 == 0:

return (nums[n // 2 - 1] + nums[n // 2]) / 2

else:

return nums[n // 2]

**5.longest palindromic substring**

Class solution:

def longestPalindrome(self, s: str) -> str:

def expandAroundCenter(left, right):

while left >= 0 and right < len(s) and s[left] == s[right]:

left -= 1

right += 1

return s[left + 1:right]

if len(s) < 1:

return ""

longest = ""

for i in range(len(s)):

odd\_palindrome = expandAroundCenter(i, i)

even\_palindrome = expandAroundCenter(i, i + 1)

longest = max(longest, odd\_palindrome, even\_palindrome, key=len)

return longest

s = "babad"

solution = Solution()

print(solution.longestPalindrome(s))

**6.zigzag conversation**

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)

s = "PAYPALISHIRING"

numRows = 3

output = convert(s, numRows)

print(output)

**7.reverse integer**

def reverse\_number(number):

reversed\_number = 0

while number > 0:

remainder = number % 10

reversed\_number = (reversed\_number \* 10) + remainder

number = number // 10

return reversed\_number

number = 123

reversed\_number = reverse\_number(number)

print(f"The reversed number of {number} is {reversed\_number}")

**8.string to integer**

num\_str = "123"

num\_int = int(num\_str)

print(num\_int)

**9.palindrome number**

def is\_palindrome(num):

return str(num) == str(num)[::-1]

num = 121

if is\_palindrome(num):

print(f"{num} is a palindrome number.")

else:

print(f"{num} is not a palindrome number.")

**10.regular expression matching**

class Solution:

def isMatch(self, s: str, p: str) -> bool:

if not p:

return not s

first\_match = bool(s) and p[0] in {s[0], '.'}

if len(p) > 1 and p[1] == '\*':

return (self.isMatch(s, p[2:]) or

first\_match and self.isMatch(s[1:], p))

else:

return first\_match and self.isMatch(s[1:], p[1:])

solution = Solution()

s = "aa"

p = "a"

print(solution.isMatch(s, p))