ASSIGNMENT-1

Name: P.SISINDRI

Rg.no : 192324122

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. You can return the answer in any order.

```python
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
    return None

nums = [2, 7, 11, 15]
target = 9
print(two_sum(nums, target))
```

Output:
```
[0, 1]

=== Code Execution Successful ===
```

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.

```python
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
```

```python
        while l1 or l2 or carry:
            val1, val2 = (l1.val if l1 else 0), (l2.val if l2 else 0)
            carry, out = divmod(val1 + val2 + carry, 10)
            current.next = ListNode(out)
            current = current.next
            l1, l2 = (l1.next if l1 else None), (l2.next if l2 else None)
        return dummy.next
def create_linked_list(lst):
    dummy = ListNode()
    current = dummy
    for number in lst:
        current.next = ListNode(number)
        current = current.next
    return dummy.next
def linked_list_to_list(node):
    result = []
    while node:
        result.append(node.val)
        node = node.next
    return result
l1 = create_linked_list([2, 4, 3])
l2 = create_linked_list([5, 6, 4])
result = addTwoNumbers(l1, l2)
print(linked_list_to_list(result))
```

```
1   class ListNode:
2       def __init__(self, val=0, next=None):
3           self.val = val
4           self.next = next
5   def addTwoNumbers(l1, l2):
6       dummy = ListNode()
7       current, carry = dummy, 0
8       while l1 or l2 or carry:
9           val1, val2 = (l1.val if l1 else 0), (l2.val if l2 else 0)
10          carry, out = divmod(val1 + val2 + carry, 10)
11          current.next = ListNode(out)
12          current = current.next
13          l1, l2 = (l1.next if l1 else None), (l2.next if l2 else None)
14      return dummy.next
15  def create_linked_list(lst):
16      dummy = ListNode()
17      current = dummy
18      for number in lst:
19          current.next = ListNode(number)
20          current = current.next
21      return dummy.next
22  def linked_list_to_list(node):
23      result = []
24      while node:
```

Output:

```
[7, 0, 8]

=== Code Execution Successful ===
```

3. Longest Substring without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

```
1   def length_of_longest_substring(s):
2       char_set = set()
3       left = 0
4       max_length = 0
5       for right in range(len(s)):
6           while s[right] in char_set:
7               char_set.remove(s[left])
8               left += 1
9           char_set.add(s[right])
10          max_length = max(max_length, right - left + 1)
11      return max_length
12  s = "abcabcbb"
13  print(length_of_longest_substring(s))
14
```

Output:

```
3

=== Code Execution Successful ===
```

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)).

```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    half_len = (m + n + 1) // 2
    imin, imax = 0, m
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j-1]:
            imin = i + 1
        elif i > 0 and nums1[i-1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = nums2[j-1]
            elif j == 0: max_of_left = nums1[i-1]
            else: max_of_left = max(nums1[i-1], nums2[j-1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m: min_of_right = nums2[j]
            elif j == n: min_of_right = nums1[i]
            else: min_of_right = min(nums1[i], nums2[j])
            return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```

Output
```
3

=== Code Execution Successful ===
```

5. Longest Palindromic Substring

Given a string s, return the longest palindromic substring in s.

```python
def longest_palindromic_substring(s):
    if len(s) == 0:
        return ""
    def expand_around_center(s, left, right):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return left + 1, right - 1
    start, end = 0, 0
    for i in range(len(s)):
        left1, right1 = expand_around_center(s, i, i)
        left2, right2 = expand_around_center(s, i, i + 1)
        if right1 - left1 > end - start:
            start, end = left1, right1
        if right2 - left2 > end - start:
            start, end = left2, right2
    return s[start:end + 1]
s = "babad"
print(longest_palindromic_substring(s))
```

Output
```
bab

=== Code Execution Successful ===
```

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);

```python
1  def convert(s, numRows):
2      if numRows == 1 or numRows >= len(s):
3          return s
4
5      rows = [''] * numRows
6      current_row, step = 0, -1
7
8      for char in s:
9          rows[current_row] += char
10         if current_row == 0 or current_row == numRows - 1:
11             step = -step
12         current_row += step
13
14     return ''.join(rows)
15
16  s = "PAYPALISHIRING"
17  numRows = 3
18  print(convert(s, numRows))
19
```

Output:
```
PAHNAPLSIIGYIR

=== Code Execution Successful ===
```

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).

```python
1  def reverse(x):
2      INT_MIN, INT_MAX = -2**31, 2**31 - 1
3      result = 0
4      sign = -1 if x < 0 else 1
5      x *= sign
6      while x:
7          digit = x % 10
8          x //= 10
9          if result > (INT_MAX - digit) // 10:
10             return 0
11         result = result * 10 + digit
12     return sign * result
13  x = 123
14  print(reverse(x))
15  x = -123
16  print(reverse(x))
17
18
```

Output:
```
321
-321

=== Code Execution Successful ===
```

8. String to Integer

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer

```
main.py                                    Save   Run        Output

 1 ▾ def myAtoi(s):                                          42
 2       INT_MIN, INT_MAX = -2**31, 2**31 - 1               -42
 3       i, n = 0, len(s)                                    4193
 4 ▾    while i < n and s[i].isspace():
 5           i += 1                                          === Code Execution Successful ===
 6       sign = 1
 7 ▾    if i < n and s[i] in ('+', '-'):
 8           sign = -1 if s[i] == '-' else 1
 9           i += 1
10       result = 0
11 ▾    while i < n and s[i].isdigit():
12           digit = int(s[i])
13 ▾        if result > (INT_MAX - digit) // 10:
14               return INT_MAX if sign == 1 else INT_MIN
15           result = result * 10 + digit
16           i += 1
17       return sign * result
18   print(myAtoi("42"))
19   print(myAtoi("   -42"))
20   print(myAtoi("4193 with words"))
21
22
```

9. Palindrome Number

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

```
main.py                                    Save   Run        Output

 1 ▾ def isPalindrome(x):                                    True
 2 ▾    if x < 0:                                            False
 3           return False
 4 ▾    if x % 10 == 0 and x != 0:                           === Code Execution Successful ===
 5           return False
 6       reverted_half = 0
 7 ▾    while x > reverted_half:
 8           reverted_half = reverted_half * 10 + x % 10
 9           x //= 10
10       return x == reverted_half or x == reverted_half // 10
11   print(isPalindrome(121))
12   print(isPalindrome(-121))
13
14
```

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).

```python
def isMatch(s, p):
    m, n = len(s), len(p)
    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True
    for j in range(1, n + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 2]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if p[j - 1] == '*':
                dp[i][j] = dp[i][j - 2]
                if p[j - 2] == '.' or p[j - 2] == s[i - 1]:
                    dp[i][j] = dp[i][j] or dp[i - 1][j]
            else:
                if p[j - 1] == '.' or p[j - 1] == s[i - 1]:
                    dp[i][j] = dp[i - 1][j - 1]
    return dp[m][n]
print(isMatch("aa", "a"))
print(isMatch("aa", "a*"))
print(isMatch("ab", ".*"))
```

Output

```
False
True
True

=== Code Execution Successful ===
```