

ASSIGNMENT-1

K. SRAVANTHI

Rg.no : 192372127

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.

main.py	Output
<pre>1 def two_sum(nums, target): 2 num_dict = {} 3 for i, num in enumerate(nums): 4 complement = target - num 5 if complement in num_dict: 6 return [num_dict[complement], i] 7 num_dict[num] = i 8 return None 9 10 nums = [2, 7, 11, 15] 11 target = 9 12 print(two_sum(nums, target)) 13</pre>	<pre>[0, 1] === Code Execution Successful ===</pre>

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.

```
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:
        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))

```

main.py	Output
<pre> 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: </pre>	<pre> [7, 0, 8] === Code Execution Successful === </pre>

3. Longest Substring without Repeating Characters

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

main.py	Output
<pre> 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 </pre>	<pre> 3 === Code Execution Successful === </pre>

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 $O(\log(m+n))$.

main.py	Output
<pre>1 def findMedianSortedArrays(nums1, nums2): 2 if len(nums1) > len(nums2): 3 nums1, nums2 = nums2, nums1 4 m, n = len(nums1), len(nums2) 5 half_len = (m + n + 1) // 2 6 imin, imax = 0, m 7 while imin <= imax: 8 i = (imin + imax) // 2 9 j = half_len - i 10 if i < m and nums1[i] < nums2[j-1]: 11 imin = i + 1 12 elif i > 0 and nums1[i-1] > nums2[j]: 13 imax = i - 1 14 else: 15 if i == 0: max_of_left = nums2[j-1] 16 elif j == 0: max_of_left = nums1[i-1] 17 else: max_of_left = max(nums1[i-1], nums2[j-1]) 18 if (m + n) % 2 == 1: 19 return max_of_left 20 if i == m: min_of_right = nums2[j] 21 elif j == n: min_of_right = nums1[i] 22 else: min_of_right = min(nums1[i], nums2[j]) 23 return (max_of_left + min_of_right) / 2.0 24 nums1 = [1, 3] 25 nums2 = [2] 26 print(findMedianSortedArrays(nums1, nums2)) 27</pre>	<pre>3 === Code Execution Successful ===</pre>

5. Longest Palindromic Substring

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

main.py	Output
<pre>1 def longest_palindromic_substring(s): 2 if len(s) == 0: 3 return "" 4 def expand_around_center(s, left, right): 5 while left >= 0 and right < len(s) and s[left] == s[right]: 6 left -= 1 7 right += 1 8 return left + 1, right - 1 9 start, end = 0, 0 10 for i in range(len(s)): 11 left1, right1 = expand_around_center(s, i, i) 12 left2, right2 = expand_around_center(s, i, i + 1) 13 if right1 - left1 > end - start: 14 start, end = left1, right1 15 if right2 - left2 > end - start: 16 start, end = left2, right2 17 return s[start:end + 1] 18 s = "babad" 19 print(longest_palindromic_substring(s)) 20</pre>	<pre>bab === Code Execution Successful ===</pre>

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);`

main.py	Output
<pre>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</pre>	<pre>PAHNAPLSIIGYIR === Code Execution Successful ===</pre>

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 $[-2^{31}, 2^{31} - 1]$, then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

main.py	Output
<pre>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</pre>	<pre>321 -321 === Code Execution Successful ===</pre>

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
<pre> 1 def myAtoi(s): 2 INT_MIN, INT_MAX = -2**31, 2**31 - 1 3 i, n = 0, len(s) 4 while i < n and s[i].isspace(): 5 i += 1 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 </pre>			<pre> 42 -42 4193 === Code Execution Successful === </pre>

9. Palindrome Number

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

main.py	Save	Run	Output
<pre> 1 def isPalindrome(x): 2 if x < 0: 3 return False 4 if x % 10 == 0 and x != 0: 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 </pre>			<pre> True False === Code Execution Successful === </pre>

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).

main.py	Save	Run	Output
<pre> 1 def isMatch(s, p): 2 m, n = len(s), len(p) 3 dp = [[False] * (n + 1) for _ in range(m + 1)] 4 dp[0][0] = True 5 for j in range(1, n + 1): 6 if p[j - 1] == '*': 7 dp[0][j] = dp[0][j - 2] 8 for i in range(1, m + 1): 9 for j in range(1, n + 1): 10 if p[j - 1] == '*': 11 dp[i][j] = dp[i][j - 2] 12 if p[j - 2] == '.' or p[j - 2] == s[i - 1]: 13 dp[i][j] = dp[i][j] or dp[i - 1][j] 14 else: 15 if p[j - 1] == '.' or p[j - 1] == s[i - 1]: 16 dp[i][j] = dp[i - 1][j - 1] 17 return dp[m][n] 18 print(isMatch("aa", "a")) 19 print(isMatch("aa", "a*")) 20 print(isMatch("ab", ".*")) 21 22 </pre>			<pre> False True True === Code Execution Successful === </pre>