

ASSIGNMENT-5

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_to_index = {} 3 for i, num in enumerate(nums): 4 complement = target - num 5 if complement in num_to_index: 6 return [num_to_index[complement], i] 7 num_to_index[num] = i 8 nums = [2, 7, 11, 15] 9 target = 9 10 print(two_sum(nums, target)) 11</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.

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 add_two_numbers(l1, l2): 6 dummy = ListNode() 7 current = dummy 8 carry = 0 9 while l1 or l2 or carry: 10 val1 = l1.val if l1 else 0 11 val2 = l2.val if l2 else 0 12 total = val1 + val2 + carry 13 carry = total // 10 14 total = total % 10 15 current.next = ListNode(total) 16 current = current.next 17 if l1: l1 = l1.next 18 if l2: l2 = l2.next 19 return dummy.next 20 def print_linked_list(node): 21 while node: 22 print(node.val, end=' ') 23 node = node.next 24 print() 25 l1 = ListNode(2, ListNode(4, ListNode(3))) 26 l2 = ListNode(5, ListNode(6, ListNode(4))) 27 result = add_two_numbers(l1, l2) 28 print_linked_list(result) 29</pre>	<pre>[0, 1] === 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_index = {} 3 max_length = 0 4 left = 0 5 for right in range(len(s)): 6 if s[right] in char_index and char_index[s[right]] >= left: 7 left = char_index[s[right]] + 1 8 char_index[s[right]] = right 9 max_length = max(max_length, right - left + 1) 10 return max_length 11 s = "abcabcbb" 12 print(length_of_longest_substring(s)) 13</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 imin, imax, half_len = 0, m, (m + n + 1) // 2 6 while imin <= imax: 7 i = (imin + imax) // 2 8 j = half_len - i 9 if i < m and nums1[i] < nums2[j - 1]: 10 imin = i + 1 11 elif i > 0 and nums1[i - 1] > nums2[j]: 12 imax = i - 1 13 else: 14 if i == 0: 15 max_of_left = nums2[j - 1] 16 elif j == 0: 17 max_of_left = nums1[i - 1] 18 else: 19 max_of_left = max(nums1[i - 1], nums2[j - 1]) 20 if (m + n) % 2 == 1: 21 return max_of_left 22 if i == m: 23 min_of_right = nums2[j] 24 elif j == n: 25 min_of_right = nums1[i] 26 else: 27 min_of_right = min(nums1[i], nums2[j]) 28 return (max_of_left + min_of_right) / 2.0 29 30 nums1 = [1, 3] 31 nums2 = [2] 32 print(findMedianSortedArrays(nums1, nums2))</pre>	<pre>2 === 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_palindrome(s): 2 def expand_around_center(left, right): 3 while left >= 0 and right < len(s) and s[left] == s[right]: 4 left -= 1 5 right += 1 6 return left + 1, right - 1 7 start, end = 0, 0 8 for i in range(len(s)): 9 left1, right1 = expand_around_center(i, i) 10 left2, right2 = expand_around_center(i, i + 1) 11 if right1 - left1 > end - start: 12 start, end = left1, right1 13 if right2 - left2 > end - start: 14 start, end = left2, right2 15 return s[start:end + 1] 16 s = "babad" 17 print(longest_palindrome(s)) 18 </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 rows = [''] * numRows 5 current_row = 0 6 going_down = False 7 for char in s: 8 rows[current_row] += char 9 if current_row == 0 or current_row == numRows - 1: 10 going_down = not going_down 11 current_row += 1 if going_down else -1 12 return ''.join(rows) 13 s = "PAYPALISHIRING" 14 numRows = 3 15 print(convert(s, numRows)) 16 </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	Save	Run	Output
<pre>1 def reverse(x): 2 INT_MIN = -2**31 3 INT_MAX = 2**31 - 1 4 is_negative = x < 0 5 if is_negative: 6 x = -x 7 result = 0 8 while x != 0: 9 digit = x % 10 10 x //= 10 11 result = result * 10 + digit 12 if result < INT_MIN or result > INT_MAX: 13 return 0 14 if is_negative: 15 result = -result 16 return result 17 x = 123 18 print(reverse(x)) 19</pre>			<pre>321 === Code Execution Successful ===</pre>

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).

The algorithm for `myAtoi(string s)` is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31} , and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result. Note:
 - Only the space character ' ' is considered a whitespace character.
 - Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.

main.py	Output
<pre>1 def myAtoi(s: str) -> int: 2 INT_MAX = 2**31 - 1 3 INT_MIN = -2**31 4 i = 0 5 while i < len(s) and s[i] == ' ': 6 i += 1 7 sign = 1 8 if i < len(s) and (s[i] == '+' or s[i] == '-'): 9 if s[i] == '-': 10 sign = -1 11 i += 1 12 num = 0 13 while i < len(s) and s[i].isdigit(): 14 num = num * 10 + int(s[i]) 15 i += 1 16 num *= sign 17 if num > INT_MAX: 18 return INT_MAX 19 elif num < INT_MIN: 20 return INT_MIN 21 else: 22 return num 23 s = "42" 24 print(myAtoi(s))</pre>	<pre>42 === Code Execution Successful ===</pre>

9. Palindrome Number

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

main.py	Output
<pre>1 def isPalindrome(x: int) -> bool: 2 str_x = str(x) 3 return str_x == str_x[::-1] 4 x = 121 5 print(isPalindrome(x)) 6 7</pre>	<pre>True === 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	Output
<pre> 1 def isMatch(s: str, p: str) -> bool: 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(2, 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] == '.' or p[j - 1] == s[i - 1]: 11 dp[i][j] = dp[i - 1][j - 1] 12 elif p[j - 1] == '*': 13 dp[i][j] = dp[i][j - 2] 14 if p[j - 2] == '.' or p[j - 2] == s[i - 1]: 15 dp[i][j] = dp[i][j] or dp[i - 1][j] 16 return dp[m][n] 17 s = "aa" 18 p = "a" 19 print(isMatch(s, p)) 20 21 </pre>	False

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 `i`th 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.

main.py	Output
<pre> 1 def maxArea(height): 2 left = 0 3 right = len(height) - 1 4 max_area = 0 5 while left < right: 6 h = min(height[left], height[right]) 7 w = right - left 8 area = h * w 9 max_area = max(max_area, area) 10 if height[left] < height[right]: 11 left += 1 12 else: 13 right -= 1 14 return max_area 15 height = [1,8,6,2,5,4,8,3,7] 16 print(maxArea(height)) 17 </pre>	49 === Code Execution Successful ===

12. Integer to Roman

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

main.py

Save

Run

```

1 def romanToInt(s: str) -> int:
2     roman_map = {
3         'I': 1,
4         'V': 5,
5         'X': 10,
6         'L': 50,
7         'C': 100,
8         'D': 500,
9         'M': 1000
10    }
11    total = 0
12    n = len(s)
13    for i in range(n):
14        if i < n - 1 and roman_map[s[i]] < roman_map[s[i + 1]]:
15            total -= roman_map[s[i]]
16        else:
17            total += roman_map[s[i]]
18    return total
19 s = "MCMXCIV"
20 print(romanToInt(s))
21

```

Output

1994

=== Code Execution Successful ===

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

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 . """

main.py	Output
<pre> 1 def romanToInt(s: str) -> int: 2 roman_map = { 3 'I': 1, 4 'V': 5, 5 'X': 10, 6 'L': 50, 7 'C': 100, 8 'D': 500, 9 'M': 1000 10 } 11 total = 0 12 n = len(s) 13 for i in range(n): 14 if i < n - 1 and roman_map[s[i]] < roman_map[s[i + 1]]: 15 total -= roman_map[s[i]] 16 else: 17 total += roman_map[s[i]] 18 return total 19 s = "MCMXCIV" 20 print(romanToInt(s)) 21 </pre>	<pre> 1994 === Code Execution Successful === </pre>

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.

main.py	Output
<pre> 1 def threeSum(nums): 2 nums.sort() 3 result = [] 4 n = len(nums) 5 for i in range(n): 6 if i > 0 and nums[i] == nums[i - 1]: 7 continue 8 left, right = i + 1, n - 1 9 while left < right: 10 total = nums[i] + nums[left] + nums[right] 11 if total == 0: 12 result.append([nums[i], nums[left], nums[right]]) 13 while left < right and nums[left] == nums[left + 1]: 14 left += 1 15 while left < right and nums[right] == nums[right - 1]: 16 right -= 1 17 left += 1 18 right -= 1 19 elif total < 0: 20 left += 1 21 else: 22 right -= 1 23 return result 24 nums = [-1,0,1,2,-1,-4] 25 print(threeSum(nums)) 26 </pre>	<pre> [[-1, -1, 2], [-1, 0, 1]] === Code Execution Successful === </pre>

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.


```
main.py  [ ] [ ] Save Run Output

1 def threeSumClosest(nums, target):
2     nums.sort()
3     closest_sum = float('inf')
4     n = len(nums)
5     for i in range(n):
6         left, right = i + 1, n - 1
7         while left < right:
8             total = nums[i] + nums[left] + nums[right]
9             if abs(total - target) < abs(closest_sum - target):
10                 closest_sum = total
11             if total < target:
12                 left += 1
13             elif total > target:
14                 right -= 1
15             else:
16                 return target
17     return closest_sum
18 nums = [-1, 2, 1, -4]
19 target = 1
20 print(threeSumClosest(nums, target))
21
```

```
2
=== Code Execution Successful ===
```

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.



```
main.py  [ ] [ ] Save Run Output

1 def letterCombinations(digits: str):
2     if not digits:
3         return []
4     mapping = {
5         '2': 'abc',
6         '3': 'def',
7         '4': 'ghi',
8         '5': 'jkl',
9         '6': 'mno',
10        '7': 'pqrs',
11        '8': 'tuv',
12        '9': 'wxyz'
13    }
14    def backtrack(combination, next_digits):
15        if len(next_digits) == 0:
16            result.append(combination)
17        else:
18            for letter in mapping[next_digits[0]]:
19                backtrack(combination + letter, next_digits[1:])
20    result = []
21    backtrack('', digits)
22    return result
23 digits = "23"
24 print(letterCombinations(digits))
```

```
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
=== Code Execution Successful ===
```

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: • $0 \leq a, b, c, d < n$

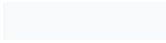
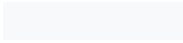
- `a, b, c,` and `d` are distinct.
- `nums[a] + nums[b] + nums[c] + nums[d] == target` You may return the answer in any order.

main.py	Output
<pre>1 def fourSum(nums, target): 2 nums.sort() 3 result = [] 4 n = len(nums) 5 for i in range(n - 3): 6 if i > 0 and nums[i] == nums[i - 1]: 7 continue 8 for j in range(i + 1, n - 2): 9 if j > i + 1 and nums[j] == nums[j - 1]: 10 continue 11 left, right = j + 1, n - 1 12 while left < right: 13 total = nums[i] + nums[j] + nums[left] + nums[right] 14 if total == target: 15 result.append([nums[i], nums[j], nums[left], nums[right]]) 16 while left < right and nums[left] == nums[left + 1]: 17 left += 1 18 while left < right and nums[right] == nums[right - 1]: 19 right -= 1 20 left += 1 21 right -= 1 22 elif total < target: 23 left += 1 24 else: 25 right -= 1 26 return result 27 nums = [1, 0, -1, 0, -2, 2] 28 target = 0 29 print(fourSum(nums, target)) 30</pre>	<pre>['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf'] === Code Execution Successful ===</pre>

19. Remove Nth Node From End of List

Given the `head` of a linked list, remove the `n`th node from the end of the list and return its head.

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 removeNthFromEnd(head, n): 6 dummy = ListNode(0) 7 dummy.next = head 8 fast = slow = dummy 9 for _ in range(n): 10 fast = fast.next 11 while fast.next: 12 fast = fast.next 13 slow = slow.next 14 slow.next = slow.next.next 15 return dummy.next 16 head = ListNode(1) 17 head.next = ListNode(2) 18 head.next.next = ListNode(3) 19 head.next.next.next = ListNode(4) 20 head.next.next.next.next = ListNode(5) 21 n = 2 22 new_head = removeNthFromEnd(head, n) 23 while new_head: 24 print(new_head.val, end=" -> ") 25 new_head = new_head.next 26</pre>	<pre>['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf'] === Code Execution Successful ===</pre>



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.

main.py	 	Save	Run	Output
<pre>1 def isValid(s: str) -> bool: 2 stack = [] 3 mapping = {"(": ")", "{": "}", "[": "]" 4 for char in s: 5 if char in mapping: 6 top_element = stack.pop() if stack else '#' 7 if mapping[char] != top_element: 8 return False 9 else: 10 stack.append(char) 11 return not stack 12 s = "()[]{}" 13 print(isValid(s)) 14</pre>	<pre>True === Code Execution Successful ===</pre>			