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 .You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

Constraints:

- **2 <= nums.length <= 104**
- **-109 <= nums[i] <= 109**
- **-109 <= target <= 109**
- **Only one valid answer exists.**

main.py

Share Run

```python
def two_sum(nums, target):
    num_to_index = {}
    for index, num in enumerate(nums):
        complement = target - num
        if complement in num_to_index:
            return [num_to_index[complement], index]
        num_to_index[num] = index
    return []


nums1 = [2, 7, 11, 15]
target1 = 9
print(two_sum(nums1, target1))
```

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.

Example 1:

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

OUTPUT:

main.py

Share Run

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def add_two_numbers(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 = l1.next if l1 else None
        l2 = l2.next if l2 else None
    return dummy.next
def to_linked_list(lst):
    head = current = ListNode()
    for number in lst:
        current.next = ListNode(number)
        current = current.next
    return head.next
def to_list(node):
    lst = []
    while node:
        lst.append(node.val)
        node = node.next
    return lst

l1 = to_linked_list([2, 4, 3])
l2 = to_linked_list([5, 6, 4])
result = add_two_numbers(l1, l2)
print(to_list(result))
```

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.

Example 1:

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

Example 3:

Input: s = "pwwkew"

Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

- 0 <= s.length <= 5 * 104
- s consists of English letters, digits, symbols and spaces.

SOURCE CODE:

main.py

```python
def length_of_longest_substring(s):
    char_set = set()
    left = 0
    max_length = 0

    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_length = max(max_length, right - left + 1)

    return max_length


print(length_of_longest_substring("abcabcbb"))
print(length_of_longest_substring("bbbbb"))
print(length_of_longest_substring("pwwkew"))
```

Output

```
3
1
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)).

Example 1: Input: nums1 = [1,3], nums2 = [2]Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2.

Example 2: Input: nums1 = [1,2], nums2 = [3,4]Output: 2.50000

Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

- nums1.length == m
- nums2.length == n
- 0 <= m <= 1000
- 0 <= n <= 1000
- 1 <= m + n <= 2000
- -106 <= nums1[i], nums2[i] <= 106

SOURCE CODE:

main.py

```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1

    m, n = len(nums1), len(nums2)
    imin, imax, half_len = 0, m, (m + n + 1) // 2

    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 float(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
print(findMedianSortedArrays([1, 3], [2]))
```

Output

```
2.0

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

5. Longest Palindromic Substring

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

Example 1:

Input: s = "babad"

Output: "bab"

Explanation: "aba" is also a valid answer.

Example 2:

Input: s = "cbbd"

Output: "bb"

Constraints:

- 1 <= s.length <= 1000
- s consist of only digits and English letters.

OUTPUT:

main.py

```python
def longest_palindromic_substring(s: str) -> str:
    n = len(s)
    if n < 2:
        return s

    start, max_length = 0, 1

    for i in range(n):
        if i - max_length >= 1 and s[i-max_length-1:i+1] == s[i-max_length-1:i+1][::-1]:
            start = i - max_length - 1
            max_length += 2
        elif i - max_length >= 0 and s[i-max_length:i+1] == s[i-max_length:i+1][::-1]:
            start = i - max_length
            max_length += 1

    return s[start:start + max_length]

print(longest_palindromic_substring("babad"))
print(longest_palindromic_substring("cbbd"))
```

Share Run

Output

```
bab
bb

=== 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);

Example 1:

Input: s = "PAYPALISHIRING", numRows = 3

Output: "PAHNAPLSIIGYIR"

Example 2:

Input: s = "PAYPALISHIRING", numRows = 4

Output: "PINALSIGYAHRPI"

Explanation:

P I N

A L S I G

Y A H R

P I

Example 3:

Input: s = "A", numRows = 1

Output: "A"

Constraints:

- 1 <= s.length <= 1000
- s consists of English letters (lower-case and upper-case), ',' and '.'.
- 1 <= numRows <= 1000

OUTPUT:

main.py

```python
def convert(s: str, numRows: int) -> str:
    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)

print(convert("PAYPALISHIRING", 3))
print(convert("PAYPALISHIRING", 4))
```

Output

```
PAHNAPLSIIGYIR
PINALSIGYAHRPI

=== 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).

Example 1: Input: x = 123 Output: 321

Example 2: Input: x = -123 Output: -321

Example 3: Input: x = 120 Output: 21

Constraints:

- -231 <= x <= 231 – 1

SOURCE CODE:

main.py

```python
n = 321;
rev = 0

while(n != 0):
    a = n % 10
    rev = rev * 10 + a
    n = n // 10

print(rev)
```

Output

```
123

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

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 [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 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.

Example 1: Input: s = "42"Output: 42

Explanation: The underlined characters are what is read in, the caret is the current reader position.

Step 1: "42" (no characters read because there is no leading whitespace)

Step 2: "42" (no characters read because there is neither a '-' nor '+')

Step 3: "42" ("42" is read in)

The parsed integer is 42. Since 42 is in the range [-231, 231 - 1], the final result is 42.

Example 2: Input: s = " -42" Output: -42

Explanation:

Step 1: " -42" (leading whitespace is read and ignored)

Step 2: " -42" ('-' is read, so the result should be negative)

Step 3: " -42" ("42" is read in)

The parsed integer is -42.

Since -42 is in the range [-231, 231 - 1], the final result is -42.

Example 3:

Input: s = "4193 with words"

Output: 4193

Explanation:

Step 1: "4193 with words" (no characters read because there is no leading whitespace)

Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')

Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)

The parsed integer is 4193.

Since 4193 is in the range [-231, 231 - 1], the final result is 4193.

Constraints:

- 0 <= s.length <= 200
- s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '-', and '.'.

SOURCE CODE:

main.py

```python
def string_to_int(s):
    try:
        return int(s)
    except ValueError:
        print("Error")
        return None
print(string_to_int("123"))
print(string_to_int("-456"))
print(string_to_int("12.34"))
```

Output

```
123
-456
Error
None

=== Code Execution Successful
```

9. Palindrome Number

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

Example 1: Input: x = 121 Output: true

Explanation: 121 reads as 121 from left to right and from right to left.

Example 2: Input: x = -121 Output: false

Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Example 3: Input: x = 10 Output: false

Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Constraints:

- -231 <= x <= 231 – 1

SOURCE CODE:

main.py

```python
def is_palindrome(x):
    if x < 0 or (x % 10 == 0 and x != 0):
        return False
    str_x = str(x)
    return str_x == str_x[::-1]
print(is_palindrome(121))
print(is_palindrome(-121))
print(is_palindrome(10))
print(is_palindrome(0))
```

Output

```
True
False
False
True

=== Code Execution Successful
```

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).

Example 1:

Input: s = "aa", p = "a"

Output: false

Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p = "a*"

Output: true

Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:

Input: s = "ab", p = ".*"

Output: true

Explanation: ".*" means "zero or more (*) of any character (.)".

Constraints:

- 1 <= s.length <= 20
- 1 <= p.length <= 30
- s contains only lowercase English letters.
- p contains only lowercase English letters, '.', and '*'.
- It is guaranteed for each appearance of the character '*', there will be a previous valid character to match.

SOURCE CODE:

main.py

```python
def isMatch(s: str, p: str) -> bool:
    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] or (dp[i - 1][j] if p[j - 2] in {s[i - 1], '.'} else False)
            else:
                dp[i][j] = dp[i - 1][j - 1] if p[j - 1] in {s[i - 1], '.'} else False

    return dp[m][n]
s = "aa"
p = "a"
print(isMatch(s, p))
```

Output

```
False

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