ASSIGNMENT-1 (03-06-24)

QUESTION 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.

PROGRAM:

```
num =[2,7,11,15]
target= 9
n = len(num)
for i in range(n-1):
  for j in range (i + 1, n):
    if num[i]+ num[j] ==target:
      print(f'target value found in [{i}, {j}]')
```

OUTPUT:

Target value found in [0, 1]

QUESTION 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.

Example 2:

Input: l1 = [0], l2 = [0]

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Constraints:

● The number of nodes in each linked list is in the range [1, 100].

● 0 <= Node.val <= 9

● It is guaranteed that the list represents a number that does not have leading zeros.

PROGRAM:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def add_two_numbers(l1, l2):
    dummy_head = ListNode()
    current = dummy_head
    carry = 0
```

```
    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = 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_head.next

def create_linked_list(lst):
    dummy_head = ListNode()
    current = dummy_head
    for number in lst:
        current.next = ListNode(number)
        current = current.next
    return dummy_head.next

def print_linked_list(node):
    numbers = []
    while node:
        numbers.append(node.val)
        node = node.next
    return numbers

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

```
print(print_linked_list(result))
```

OUTPUT:

[7, 0, 8]

QUESTION 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.

PROGRAM:

```
def length_of_longest_substring(s):
    char_index = {}
    longest = start = 0

    for i, char in enumerate(s):
        if char in char_index and char_index[char] >= start:
            start = char_index[char] + 1
        char_index[char] = i
```

```
        longest = max(longest, i - start + 1)

    return longest

s1 = "abcabcbb"
print(length_of_longest_substring(s1))
```

OUTPUT:

3

QUESTION 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

PROGRAM:

```
def find_median_sorted_arrays(nums1, nums2):
    A, B = nums1, nums2
    if len(A) > len(B):
```

```
    A, B = B, A
  m, n = len(A), len(B)
  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 A[i] < B[j-1]:
      imin = i + 1
    elif i > 0 and A[i-1] > B[j]:
      imax = i - 1
    else:
      if i == 0: max_of_left = B[j-1]
      elif j == 0: max_of_left = A[i-1]
      else: max_of_left = max(A[i-1], B[j-1])

      if (m + n) % 2 == 1:
        return max_of_left

      if i == m: min_of_right = B[j]
      elif j == n: min_of_right = A[i]
      else: min_of_right = min(A[i], B[j])

      return (max_of_left + min_of_right) / 2.0

nums1 = [1, 3]
nums2 = [2]
print(find_median_sorted_arrays(nums1, nums2))
```

OUTPUT:

2.0

QUESTION 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.

PROGRAM:

```
def longest_palindrome(s):
    if not s:
        return ""

    start = end = 0

    for i in range(len(s)):
        len1 = expand_around_center(s, i, i)
        len2 = expand_around_center(s, i, i + 1)
        max_len = max(len1, len2)

        if max_len > end - start:
            start = i - (max_len - 1) // 2
            end = i + max_len // 2

    return s[start:end + 1]

def expand_around_center(s, left, right):
    while left >= 0 and right < len(s) and s[left] == s[right]:
```

```
        left -= 1
        right += 1
    return right - left - 1

s1 = "babad"
print(longest_palindrome(s1))
```

OUTPUT:

Bab

QUESTION 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

PROGRAM:

```
def convert(s, numRows):
   if numRows == 1 or numRows >= len(s):
      return s

   rows = [''] * numRows
   cur_row = 0
   going_down = False

   for char in s:
      rows[cur_row] += char
      if cur_row == 0 or cur_row == numRows - 1:
         going_down = not going_down
      cur_row += 1 if going_down else -1

   return ''.join(rows)

s1 = "PAYPALISHIRING"
numRows1 = 3
print(convert(s1, numRows1))
```

OUTPUT:

PAHNAPLSIIGYIR

QUESTION 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

PROGRAM:

```
def reverse(x):
    sign = -1 if x < 0 else 1
    x *= sign
    rev = 0

    while x:
        rev = rev * 10 + x % 10
        x //= 10

    rev *= sign
    if rev < -2**31 or rev > 2**31 - 1:
        return 0
    return rev

x1 = 123
print(reverse(x1))
```

OUTPUT:

321

QUESTION 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 '.'.

PROGRAM:

```
def my_atoi(s):
    i = 0
    n = len(s)
    while i < n and s[i] == ' ':
        i += 1

    sign = 1
    if i < n and (s[i] == '-' or s[i] == '+'):
        sign = -1 if s[i] == '-' else 1
        i += 1
```

```
    num = 0
    while i < n and s[i].isdigit():
        num = num * 10 + int(s[i])
        i += 1

    num *= sign
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    if num < INT_MIN:
        return INT_MIN
    if num > INT_MAX:
        return INT_MAX
    return num

s1 = "42"
print(my_atoi(s1))
```

OUTPUT:

42

QUESTION 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

PROGRAM:

```
def is_palindrome(x):
    if x < 0:
        return False
    rev = 0
    temp = x
    while temp:
        rev = rev * 10 + temp % 10
        temp //= 10
    return x == rev

x1 = 121
print(is_palindrome(x1))
```

OUTPUT:

True

QUESTION 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.

PROGRAM:

```
def is_match(s, p):
    dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]
    dp[0][0] = True

    for j in range(1, len(p) + 1):
        if p[j-1] == '*':
            dp[0][j] = dp[0][j-2]

    for i in range(1, len(s) + 1):
        for j in range(1, len(p) + 1):
            if p[j-1] == s[i-1] or p[j-1] == '.':
                dp[i][j] = dp[i-1][j-1]
            elif p[j-1] == '*':
                dp[i][j] = dp[i][j-2]
```

```
            if p[j-2] == s[i-1] or p[j-2] == '.':
                dp[i][j] |= dp[i-1][j]

    return dp[-1][-1]

s1 = "aa"
p1 = "a"
print(is_match(s1, p1))
```

OUTPUT: false