1. [**Two Sum**](https://leetcode.com/problems/two-sum/)

**Statement:**

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 Code:**

class Solution(object):

    def twoSum(self, nums, target):

        hash\_map = {}

        for i, num in enumerate(nums):

            if target - num in hash\_map:

                return [hash\_map[target - num], i]

            hash\_map[num] = i

        return []

**Result**:

nums = [2,7,11,15]

target =9

Output

[0,1]

[**2. Add Two Numbers**](https://leetcode.com/problems/add-two-numbers/)

**Statement**:

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 code:**

class Solution(object):

    def addTwoNumbers(self, l1, l2):

        dummy = ListNode()   # dummy head node

        curr = dummy

        carry = 0

        while l1 or l2 or carry:

            val1 = l1.val if l1 else 0

            val2 = l2.val if l2 else 0

            total = val1 + val2 + carry

            carry = total // 10

            curr.next = ListNode(total % 10)

            curr = curr.next

            # move to next nodes

            if l1:

                l1 = l1.next

            if l2:

                l2 = l2.next

        return dummy.next

**Result:**

Input

l1 =[2,4,3]

l2 =[5,6,4]

Output

[7,0,8]

**3.Longest Substring Without Repeating Characters**

**Statement:**

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

**Python code:**

class Solution(object):

    def lengthOfLongestSubstring(self, s):

        seen = {}

        max\_length = 0

        start = 0

        for end, char in enumerate(s):

            if char in seen:

                start = max(start, seen[char] + 1)

            seen[char] = end

            max\_length = max(max\_length, end - start + 1)

        return max\_length

**Result:**

Input

s ="abcabcbb"

Output

3

**4.Median Of Two Sorted Arrays**

**Statement:**

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 code:**

class Solution(object):

    def findMedianSortedArrays(self, nums1, nums2):

        """

        :type nums1: List[int]

        :type nums2: List[int]

        :rtype: float

        """

        # Ensure nums1 is the smaller array to optimize binary search

        if len(nums1) > len(nums2):

            nums1, nums2 = nums2, nums1

        x, y = len(nums1), len(nums2)

        low, high = 0, x

        while low <= high:

            partitionX = (low + high) // 2

            partitionY = (x + y + 1) // 2 - partitionX

            # If partitionX is 0 it means nothing is there on left side. Use -inf for maxLeftX

            maxLeftX = float('-inf') if partitionX == 0 else nums1[partitionX - 1]

            minRightX = float('inf') if partitionX == x else nums1[partitionX]

            maxLeftY = float('-inf') if partitionY == 0 else nums2[partitionY - 1]

            minRightY = float('inf') if partitionY == y else nums2[partitionY]

            if maxLeftX <= minRightY and maxLeftY <= minRightX:

                # Found the correct partition

                if (x + y) % 2 == 0:

                    return (max(maxLeftX, maxLeftY) + min(minRightX, minRightY)) / 2.0

                else:

                    return max(maxLeftX, maxLeftY)

            elif maxLeftX > minRightY:

                high = partitionX - 1

            else:

                low = partitionX + 1

**Result:**

Input

nums1 =[1,3]

nums2 =[2]

Output

2.00000

**5.Longest palindromic Substring**

**Statement:**

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

**Python code:**

class Solution(object):

    def longestPalindrome(self, s):

        """

        :type s: str

        :rtype: str

        """

        if not s or len(s) < 1:

            return ""

        start, end = 0, 0

        for i in range(len(s)):

            len1 = self.expandAroundCenter(s, i, i)       # Odd length

            len2 = self.expandAroundCenter(s, i, i + 1)   # Even length

            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 expandAroundCenter(self, s, left, right):

        while left >= 0 and right < len(s) and s[left] == s[right]:

            left -= 1

            right += 1

        return right - left – 1

**Result:**

Input

s ="babad"

Output

"aba"

**6.Zigzag Conversion**

**Statement:**

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"

**Python code:**

class Solution(object):

    def convert(self, s, numRows):

        if numRows == 1 or numRows >= len(s):

            return s

        rows = [''] \* numRows

        current\_row = 0

        going\_down = False

        for char in s:

            rows[current\_row] += char

            # Change direction if we are at the top or bottom

            if current\_row == 0 or current\_row == numRows - 1:

                going\_down = not going\_down

            current\_row += 1 if going\_down else -1

        return ''.join(rows)

**Result:**

Input

s ="PAYPALISHIRING"

numRows =3

Output

"PAHNAPLSIIGYIR"

[**7. Reverse Integer**](https://leetcode.com/problems/reverse-integer/)

**Statement:**

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.

**Python code:**

class Solution(object):

    def reverse(self, x):

        sign = -1 if x < 0 else 1

        x = abs(x)

        rev = 0

        while x != 0:

            digit = x % 10

            rev = rev \* 10 + digit

            x //= 10

        rev \*= sign

        # 32-bit integer overflow check

        if rev < -2\*\*31 or rev > 2\*\*31 - 1:

            return 0

        return rev

**Result:**

Input

x =123

Output

321

**8.String To Integer**

**Statement:**

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

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

1. **Whitespace**: Ignore any leading whitespace (" ").
2. **Signedness**: Determine the sign by checking if the next character is '-' or '+', assuming positivity if neither present.
3. **Conversion**: Read the integer by skipping leading zeros until a non-digit character is encountered or the end of the string is reached. If no digits were read, then the result is 0.
4. **Rounding**: If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then round the integer to remain in the range. Specifically, integers less than -231 should be rounded to -231, and integers greater than 231 - 1 should be rounded to 231 - 1.

Return the integer as the final result.

**Python code:**

class Solution(object):

    def myAtoi(self, s):

        """

        :type s: str

        :rtype: int

        """

        s = s.lstrip()  # Remove leading whitespaces

        if not s:

            return 0

        sign = 1

        index = 0

        result = 0

        INT\_MAX = 2\*\*31 - 1

        INT\_MIN = -2\*\*31

        # Handle optional sign

        if s[index] == '+':

            index += 1

        elif s[index] == '-':

            sign = -1

            index += 1

        # Convert digits to integer

        while index < len(s) and s[index].isdigit():

            digit = int(s[index])

            # Check for overflow/underflow before adding digit

            if result > (INT\_MAX - digit) // 10:

                return INT\_MAX if sign == 1 else INT\_MIN

            result = result \* 10 + digit

            index += 1

        return sign \* result

**Result:**

Input

s ="42"

Output

42

**9.Palindrome Number**

**Statement:**

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

**Python code:**

class Solution:

     def isPalindrome(self, x):

        # Negative numbers and numbers ending with 0 (but not 0 itself)

        if x < 0 or (x % 10 == 0 and x != 0):

            return False

        reversed\_num = 0

        while x > reversed\_num:

            digit = x % 10

            reversed\_num = reversed\_num \* 10 + digit

            x //= 10

        # For odd length numbers, ignore the middle digit (reversed\_num // 10)

        return x == reversed\_num or x == reversed\_num // 10

**Result:**

Input

x =121

Output

True

**10.Regular Expression Matching**

**Statement:**

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 code**

class Solution(object):

    def isMatch(self, s, p):

        """

        :type s: str

        :type p: str

        :rtype: bool

        """

        m, n = len(s), len(p)

        # dp[i][j] means: s[0..i-1] matches p[0..j-1]

        dp = [[False] \* (n + 1) for \_ in range(m + 1)]

        dp[0][0] = True  # empty string matches empty pattern

        # Handle patterns like a\*, a\*b\*, a\*b\*c\* that can match empty string

        for j in range(2, 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] == '.' or p[j - 1] == s[i - 1]:

                    # Match current character

                    dp[i][j] = dp[i - 1][j - 1]

                elif p[j - 1] == '\*':

                    # Two cases: zero occurrence or at least one

                    dp[i][j] = dp[i][j - 2]  # zero occurrence

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

                        dp[i][j] |= dp[i - 1][j]  # one or more occurrences

        return dp[m][n]

**Result:**

Input

s ="aa"

p ="a"

Output

false