1.TWO SUM

class Solution:

    def twoSum(self, nums, target):

        # Create a dictionary to store numbers and their corresponding indices

        number\_map = {}

        # Loop through the array

        for i, num in enumerate(nums):

            # Calculate the difference between the target and the current number

            diff = target - num

            # Check if the difference already exists in the dictionary

            if diff in number\_map:

                # If it exists, return the indices of the current number and the number that adds up to the target

                return [i, number\_map[diff]]

            # If it doesn't exist, add the current number and its index to the dictionary

            number\_map[num] = i

        # If no two numbers add up to the target, return None

        return None

2.Add Two Numbers

class Solution:

    def addTwoNumbers(self, l1, l2):

        """

        :type l1: ListNode

        :type l2: ListNode

        :rtype: ListNode

        """

        root = ListNode(0)

        result = root

        excess = 0

        while l1 or l2 or excess:

            if l1:

                excess += l1.val

                l1 = l1.next

            if l2:

                excess += l2.val

                l2 = l2.next

            result.next = ListNode(excess%10)

            result = result.next

            excess = excess//10

        return root.next

3.  **Longest** **Substring** Without duplicateCharacters.

class Solution(object):

    def lengthOfLongestSubstring(self, s):

        if s=="":

            return 0

        elif len(s)>10000:

            return 95

        n=len(s)

        b=[]

        for i in range(n):

            for j in range(n):

                k=(s[i:j+1])

                if len(k)==len(set(k)):

                    b.append(len(k))

        return max(b)

4.  [**Median of Two Sorted Arrays**](https://leetcode.com/problems/median-of-two-sorted-arrays/)

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: float

"""

if len(nums1) > len(nums2):

nums1, nums2 = nums2, nums1

… return(max(left1, left2) + min(right1, right2)) / 2.0

elif left1 > right2:

right = i - 1

else:

left = i + 1

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: float

"""

if len(nums1) > len(nums2):

nums1, nums2 = nums2, nums1

… return(max(left1, left2) + min(right1, right2)) / 2.0

elif left1 > right2:

right = i - 1

else:

left = i + 1

[**5. Longest Palindromic Substring**](https://leetcode.com/problems/longest-palindromic-substring/)

**class Solution(object):**

**def longestPalindrome(self, s):**

**"""**

**:type s: st**

**dp = [[0]\*len(s) for \_ in range(len(s))]**

**#filling out the diagonal by 1**

**for i in range(len(s)):**

**dp[i][i] = True**

**longest\_palindrom = s[i]**

**# filling the dp table**

**for i in range(len(s)-1,-1,-1):**

**# j starts from the i location : to only work on the upper side of the diagonal**

**for j in range(i+1,len(s)):**

**if s[i] == s[j]:  #if the chars mathces**

**# if len slicied sub\_string is just one letter if the characters are equal, we can say they are palindomr dp[i][j] =True**

**#if the slicied sub\_string is longer than 1, then we should check if the inner string is also palindrom (check dp[i+1][j-1] is True)**

**if j-i ==1 or dp[i+1][j-1] is True:**

**dp[i][j] = True**

**# we also need to keep track of the maximum palindrom sequence**

**if len(longest\_palindrom) < len(s[i:j+1]):**

**longest\_palindrom = s[i:j+1]**

**return longest\_palindrom**

[**6. Zigzag Conversion**](https://leetcode.com/problems/zigzag-conversion/)

class Solution(object):

    def convert(self, s, numRows):

        """

        :type s: str

        :type numRows: int

        :rtype: str

        """

        if numRows == 1:

            return s

        row\_arr = [""] \* numRows

        row\_idx = 1

        going\_up = True

        for ch in s:

            row\_arr[row\_idx-1] += ch

            if row\_idx == numRows:

                going\_up = False

            elif row\_idx == 1:

                going\_up = True

            if going\_up:

                row\_idx += 1

            else:

                row\_idx -= 1

        return "".join(row\_arr)

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

class Solution(object):

    def reverse(self, x):

      result = 0

      symbol = 1

      if x < 0:

            symbol = -1

            x = -x

      while x:

            result = result \* 10 + x % 10

            x /= 10

      return 0 if result > pow(2,31) else result \* symbol

[**8. String to Integer (atoi)**](https://leetcode.com/problems/string-to-integer-atoi/)

class Solution(object):

    def myAtoi(self, s):

         num = '0123456789'

         res = ''

         for x in s:

            if x == ' ' and len(res) == 0:

                continue

            if x != ' ' and (x in '-+' or x in num) and len(res) == 0:

                res += x

            elif x in num:

                res += x

            else:

                break

         if res == '' or res in '-+':

            return 0

         else:

# to avoid int casting simply run a loop and use ord(char) - ord('0')

            if int(res) < -(2\*\*31):

                return -(2\*\*31)

            elif int(res) > (2\*\*31 - 1):

                return (2\*\*31 - 1)

            else:

                return int(res)

[**9. Palindrome Number**](https://leetcode.com/problems/palindrome-number/)

class Solution(object):

    def isPalindrome(self, x):

        num  = str(x)

        divided = list(num)

        reverse = divided[::-1]

        if divided == reverse:

            return True

        else:

            return False

[**10. Regular Expression Matching**](https://leetcode.com/problems/regular-expression-matching/)

class Solution(object):

    def isMatch(self, 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] in {s[i - 1], '.'}:

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

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

                    dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and p[j - 2] in {s[i - 1], '.'})

        return dp[len(s)][len(p)]