3. Longest Substring Without Repeating Characters(1)

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.

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        l = 0
        res = 0
        hasSet = set()
        for r in range(len(s)):
            while s[r] in hasSet:
                hasSet.remove(s[l])
                l=l+1
            hasSet.add(s[r])
            res = max(res, r-l+1)
        return res
```

53. Maximum Subarray(2)

Given an integer array nums, find the subarray with the largest sum, and return *its sum*

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.

Example 3:

Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        maxSum = nums[0]
        currentSum = 0
        for n in nums :
            currentSum = currentSum+n
            maxSum = max(maxSum, currentSum)
            if currentSum<0:
                currentSum = 0
        return maxSum
```

5. Longest Palindromic Substring(3)

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

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        res = ''
        reslen = 0
        for i in range(len(s)):
            l = i
            r = i
            while l>=0 and r<len(s) and s[l]==s[r]:
                if r-l+1>reslen:
                    reslen = r-l+1
                    res = s[l:r+1]
                l = l-1
                r = r+1
            l = i
            r = i+1
            while l>=0 and r<len(s) and s[l]==s[r]:
                if r-l+1>reslen:
                    reslen = r-l+1
                    res = s[l:r+1]
                l = l-1
                r = r+1
        return res
```

125. Valid Palindrome(4)

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.Given a string s, return true *if it is a **palindrome**, or* false *otherwise*.

Example 1:

Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
Example 3:

Input: s = " "

Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.

```
class Solution:

    def isPalindrome(self, s: str) -> bool:

        l = 0

        r = len(s)-1

        while l<r:

            while l<r and not s[l].isalnum():

                l = l+1
```

```
            while l<r and not s[r].isalnum():
                r = r-1
            if s[l].lower() != s[r].lower():
                return False
            l = l+1
            r = r-1
        return True
```

9. Palindrome Number(5)

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.

```
class Solution:

    def isPalindrome(self, x: int) -> bool:

        a = str(x)

        l = 0

        r = len(a)-1

        while l<r:

            if a[l]!=a[r]:

                return False

            l = l + 1

            r = r - 1

        return True
```

8. String to Integer (atoi)(6)

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.

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 [-2^{31}, 2^{31} - 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 [-2^{31}, $2^{31} - 1$], 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 [-2^{31}, $2^{31} - 1$], the final result is 4193.

```
class Solution:

    def myAtoi(self, s: str) -> int:

        s = s.strip()

        if not s:

            return 0

        sign = 1
```

```
if s[0] in ['-', '+']:
    if s[0]=='-':
        sign = -1
        s = s[1:]
    elif s[0]=='+':
        s = s[1:]
result = 0


for c in s:
    if not c.isdigit():
        break
    result = result*10 + int(c)
result = result * sign


result = max(min(result, 2**31-1), -2**31)
return result
```

7. Reverse Integer(7)

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).

Example 1:

Input: x = 123

Output: 321

Example 2:

Input: x = -123

Output: -321

Example 3:

Input: x = 120

Output: 21

```
class Solution:

    def reverse(self, x: int) -> int:

        sign = 1

        if x<0:

            sign = -1

            x = -x

        reversedNum = str(x)[::-1]

        reverseValue = int(reversedNum)*sign
```

```
if reverseValue>2**31-1 or reverseValue<-2**31:
    return 0
else:
    return reverseValue
```