Roman to Integer

Problem Definition

Roman numerals are represented by seven different symbols: `I, V, X, L, C, D and M`.

```
Symbol       Value
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

For example, `2` is written as `II` in Roman numeral, just two ones added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example Cases

Example 1:

Input: s = "III"
Output: 3
Explanation: III = 3.

Example 2:

Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.

Example 3:

Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Constraints

Constraints:

- 1 <= s.length <= 15
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is guaranteed that s is a valid roman numeral in the range [1, 3999].

Example test cases

In [1]:
```
def test_cases():
    assert romanToInt("III") == 3
    assert romanToInt("LVIII") == 58
    assert romanToInt("MCMXCIV") == 1994
    print("All test cases passed!")
```

Solutions

$O(n)$

The solution that came up first to my mind was to iterate over the letters and check the next item.

If the next letter has a bigger value in the mapping than the current one, then:

- add to the result the difference and skip the next letter

else:

- add the current letter's value.

In [4]:
```
mapping = {
    "I": 1,
    "V": 5,
    "X": 10,
    "L": 50,
    "C": 100,
    "D": 500,
    "M": 1000
}
```

In [49]:
```
def romanToInt(s):
    res = 0
    pointer = 0
    while pointer < len(s):
        curr_el = mapping[s[pointer]]
        next_el = mapping[s[pointer+1]] if (len(s) - pointer) > 1 else 0
        if next_el > curr_el:
            res += (next_el - curr_el)
            pointer += 2
        else:
            res += curr_el
            pointer += 1
    return res
```

In [48]:
```
test_cases()
```

```
All test cases passed!
```

$O(n)$

Same can be written in the reverse, which can be more readable and efficient removing the need for extra condition check.

In [54]:
```
def romanToInt(s):
    res = 0
    start = 0
    for el in reversed(s):
        if mapping[el] < start:
            res -= mapping[el]
        else:
            res += mapping[el]
        start = mapping[el]
    return res
```

In [55]:
```
test_cases()
```

```
All test cases passed!
```