Detailed Explanation of Python Code for Roman Numeral Conversion

Diana Akolzina

December 29, 2023

1 Introduction

This document provides an in-depth explanation of the Python code for converting Roman numerals to integers and vice versa, focusing on the data structures, algorithms, and programming constructs used.

2 The RomanNumerals Class

The code defines a class named `RomanNumerals`, encapsulating the functionality for Roman numeral conversion.

2.1 Mapping Dictionary

The class contains a dictionary named `roman_to_int_mapping` for mapping Roman numeral characters to integers.

```
roman_to_int_mapping = {
    'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000
}
```

Detailed Explanation:

- The dictionary uses characters as keys (e.g., 'I', 'V') and their corresponding integer values as dictionary values.
- This mapping is created for the conversion process as it provides a quick lookup to translate Roman numerals to their integer equivalents.

2.2 Method: roman_to_int

Converts a Roman numeral string to an integer.

```
def roman_to_int(self, s: str) -> int:
    total = 0
    prev_value = 0
    for char in reversed(s):
        value = self.roman_to_int_mapping[char]
        if value < prev_value:
            total -= value
        else:
            total += value
        prev_value = value
    return total
```

Detailed Explanation:

1. Initialize `total` and `prev_value` to 0. These variables will keep track of the cumulative total and the previously processed numeral's value, respectively.
2. Loop through the string `s` in reverse. This reverse iteration is crucial for handling Roman numerals' subtraction rule.
3. For each character `char` in `s`, find its corresponding integer value from `roman_to_int_mapping`.
4. If the current value is less than `prev_value`, subtract it from `total`, indicating a subtraction scenario in Roman numerals.
5. If the current value is greater than or equal to `prev_value`, add it to `total`.
6. Update `prev_value` to the current value for the next iteration.
7. After the loop, return the `total`, which now holds the integer equivalent of the Roman numeral.

2.3 Method: int_to_roman

Converts an integer to a Roman numeral string.

```
def int_to_roman(self, num: int) -> str:
    value_symbols = [
        (1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),
        (100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),
        (10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')
    ]
    roman_numeral = ''
    for value, symbol in value_symbols:
        while num >= value:
            roman_numeral += symbol
            num -= value
    return roman_numeral
```

Detailed Explanation:

1. Initialize a list of tuples `value_symbols` containing pairs of integers and their corresponding Roman numerals in descending order.
2. The list covers all the basic numerals and their common subtractive forms (like 'IV' for 4, 'IX' for 9, etc.).
3. Initialize an empty string `roman_numeral` to build the resulting Roman numeral.
4. Iterate through each tuple in `value_symbols`. For each tuple:
    (a) Check if `num` is greater than or equal to the tuple's integer value.
    (b) If true, append the tuple's Roman numeral to `roman_numeral`.
    (c) Subtract the tuple's integer value from `num`.
    (d) Repeat this process until `num` is less than the tuple's integer value.
5. Return the string `roman_numeral`, which is the Roman numeral representation of the input integer.

3 Example Usage

Demonstration of the class functionality with an example.

```
converter = RomanNumerals()
print(converter.roman_to_int("MCMXCIV"))   # Output: 1994
print(converter.int_to_roman(1994))          # Output: MCMXCIV
```

Explanation:

- An instance of `RomanNumerals` is created.
- The `roman_to_int` method is called with "MCMXCIV", and the output is 1994.
- The `int_to_roman` method is called with 1994, and the output is "MCMXCIV".