

지난 주까지의 진척사항

- 사칙연산이 가능한 계산기의 완성
- UI 구성 코드를 개선
 - 반복을 가급적 줄이는 방향으로 코드를 개선
 - 그 결과로 상수/함수 버튼의 추가가 쉽게 이루어질 수 있었음
- 상수/함수 버튼의 기능 구현 코드 개선
 - 눌러는 버튼에 따라 표시될 내용 또는 호출될 함수를 연결하는 자료 구조를 착안
- 예외 처리
 - 계산이 불가능한 경우 등 에러 발생할 여지를 가급적 철저하게 고려

이번 주의 코드 개발 내용

- 10진수 → 로마숫자 변환 알고리즘 개발
 - 몇 단계에 걸쳐서 코드 개선
- 로마숫자 → 10진수 변환 알고리즘 개발
 - 위 고찰을 토대로, 과제



우선은 버튼을 추가

functions.py

```
from calcFunctions import *
```

```
functionMap = [  
    ('factorial (!)', factorial),  
    ('-> binary', decToBin),  
    ('binary -> dec', binToDec),  
    ('-> roman', decToRoman),  
    ('roman -> dec', romanToDec),  
]
```

```
functionList = [x[0] for x in functionMap]
```

calcFunctions.py


```
def factorial(numStr):
```

```
    ...
```


```
def ...
```

```
def decToRoman(numStr):  
    return 'to Roman'
```

```
def romanToDec(numStr):  
    return 'Roman to Dec'
```



오늘 구현할 함수



오늘 과제

이렇게만 하면 버튼이 추가되나요? 왜?

함수 버튼 생성 코드

```
buttonGroups = {  
    'num': {'buttons': numPadList, 'layout': numLayout, 'columns': 3},  
    'op': {'buttons': operatorList, 'layout': opLayout, 'columns': 2},  
    'constants': {'buttons': constantList, 'layout': constLayout, 'columns': 1},  
    'functions': {'buttons': functionList, 'layout': funcLayout, 'columns': 1},  
}
```

```
for label in buttonGroups.keys():  
    r = 0; c = 0  
    buttonPad = buttonGroups[label]  
    for btnText in buttonPad['buttons']:  
        button = Button(btnText, self.buttonClicked)  
        buttonPad['layout'].addWidget(button, r, c)  
        c += 1  
        if c >= buttonPad['columns']:  
            c = 0; r += 1
```

```
functionMap = [  
    ('factorial (!)', factorial),  
    ('-> binary', decToBin),  
    ('binary -> dec', binToDec),  
    ('-> roman', decToRoman),  
    ('roman -> dec', romanToDec),  
]
```

```
functionList = [x[0] for x in functionMap]
```

그런데, 버튼 누르면 알맞은 함수도 호출되나요? 왜?

버튼 눌림에 대한 이벤트 핸들러 코드

```
Calculator.buttonClicked()
```

```
...
```

```
elif key in functionList:
```

```
    n = self.display.text()
```

```
    value = functionMap[functionList.index(key)][1](n)
```

```
    self.display.setText(str(value))
```

먼저 코드를 작성할 때

이후의 확장을 염두에 두고 하면

나중의 일이 쉬워집니다.

```
functionMap = [  
    ('factorial (!)', factorial),  
    ('-> binary', decToBin),  
    ('binary -> dec', binToDec),  
    ('-> roman', decToRoman),  
    ('roman -> dec', romanToDec),  
]
```

```
functionList = [x[0] for x in functionMap]
```


로마 숫자 표현



위 표에 있는 문자까지만 사용하면
얼마까지의 수를 표현할 수 있나요?

수	로마자
1	I
5	V
10	X
50	L
100	C
500	D
1000	M

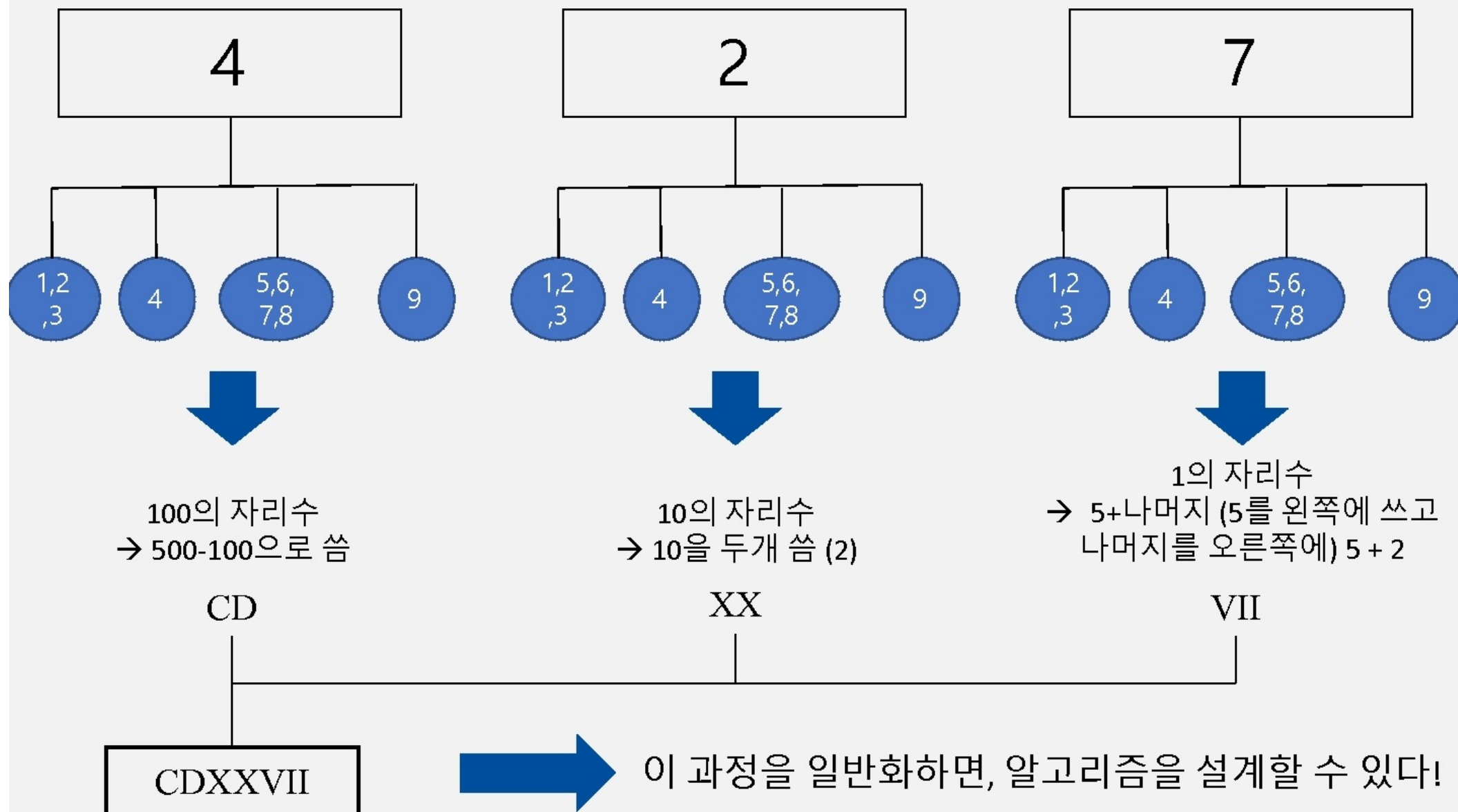
어떤 수의 오른쪽에 쓰여진 수는 더해지고,
왼쪽에 쓰여진 수는 빼진다.

3 까지는 I 를 거듭 쓴다.

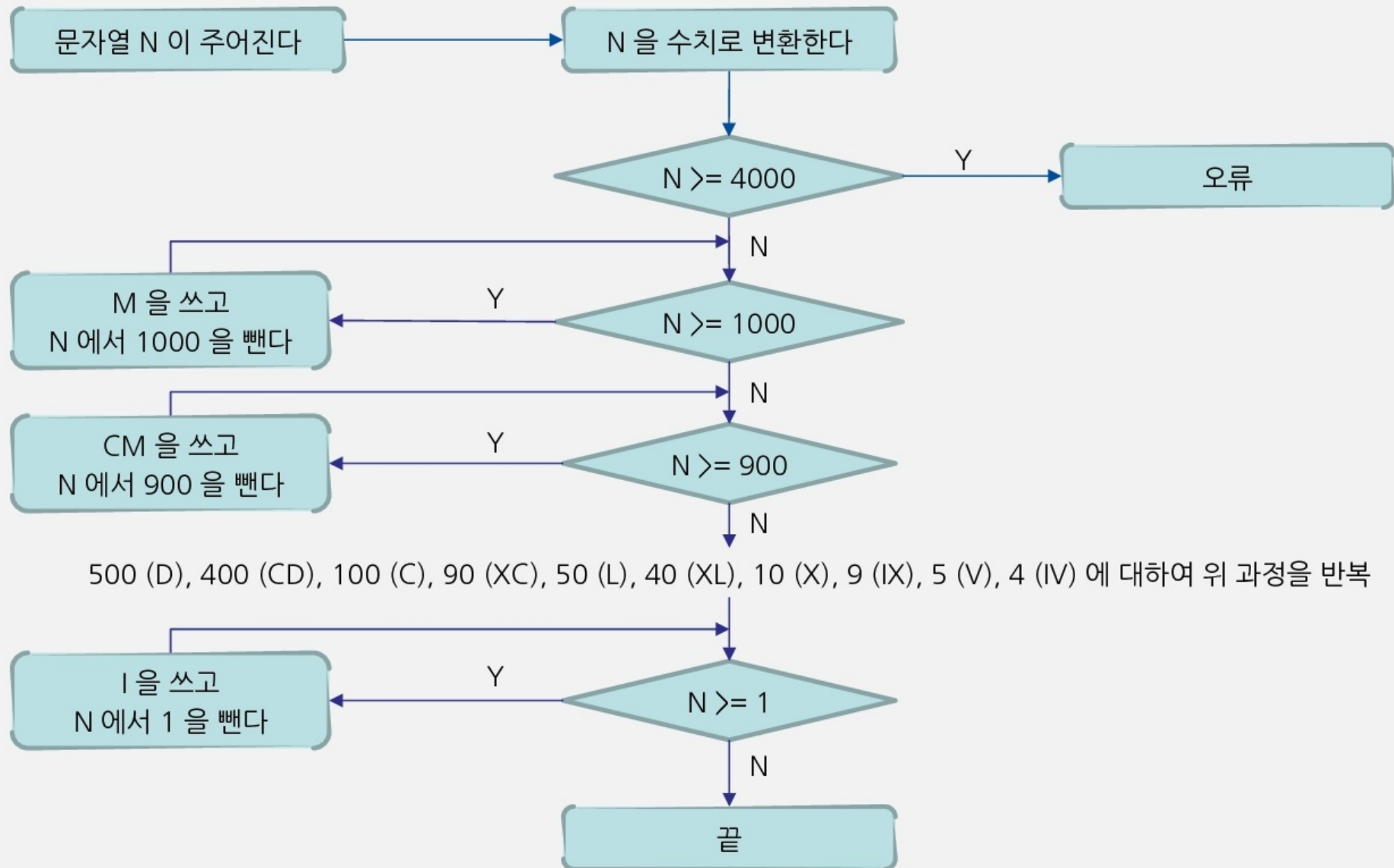
4는 5 - 1 이므로 V 의 왼쪽에 I 를 쓴다.

예: 248 = CCXLVIII

로마 숫자 쓰기 - 예제 흐름



로마 숫자 함수 구현 - 알고리즘 설계



로마 숫자 표현 함수 구현

```
def decToRoman(numStr):
```

```
    try:
```

```
        n = int(numStr)
```

```
    except:
```

```
        return 'Error!'
```

```
    if n >= 4000:
```

```
        return 'Error!'
```

```
    result = ''
```

```
    while n >= 1000:
```

```
        result += "M"
```

```
        n -= 1000
```

```
    while n >= 900:
```

```
        result += "CM"
```

```
        n -= 900
```

```
    while n >= 500:
```

```
        result += "D"
```

```
        n -= 500
```

```
    while n >= 400:
```

```
        result += "CD"
```

```
        n -= 400
```

```
    while n >= 100:
```

```
        result += "C"
```

```
        n -= 100
```

```
    while n >= 90:
```

```
        result += "XC"
```

```
        n -= 90
```

```
    while n >= 50:
```

```
        result += "L"
```

```
        n -= 50
```

```
    while n >= 40:
```

```
        result += "XL"
```

```
        n -= 40
```

```
    while n >= 10:
```

```
        result += "X"
```

```
        n -= 10
```

```
    while n >= 9:
```

```
        result += "IX"
```

```
        n -= 9
```

```
    while n >= 5:
```

```
        result += "V"
```

```
        n -= 5
```

```
    while n >= 4:
```

```
        result += "IV"
```

```
        n -= 4
```

```
    while n >= 1:
```

```
        result += "I"
```

```
        n -= 1
```

```
    return result
```

코드의 반복이 심하다.

이후에, 백만까지의 로마 숫자를 표현하도록
프로그램을 고치면 어떻게 될까?

10진수 → 로마숫자 변환: 실행 흐름 예

248 을 로마 숫자로 변환하는 예의 흐름

n = 148, result = [C] -100: C

n = 48, result = [CC] -100: C

n = 8, result = [CCXL] -40: XL

n = 3, result = [CCXLV] -5: V

n = 2, result = [CCXLVI] -1: I

n = 1, result = [CCXLVII] -1: I

n = 0, result = [CCXLVIII] -1: I

수	로마자
1	I
5	V
10	X
50	L
100	C
500	D
1000	M

로마 숫자 표현 함수 구현 개선

- 리스트와 사전을 이용해 보자!

```
if n >= 4000:  
    return 'Error!'
```

```
numberBreaks = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]  
letters = {  
    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'  
}
```

```
result = "  
for value in numberBreaks:  
    while n >= value:  
        result += letters[value]  
        n -= value  
return result
```

많이 나아진 것 같네요.

→ 하지만, 완전히 만족스러운가요?

그렇지 않다면,
무엇이 아쉬운 점인가요?

코드 개선점 착안

```
if n >= 4000:  
    return 'Error!'
```

```
numberBreaks = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]  
letters = {  
    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'  
}
```

- 불필요한 반복이 있지는 않은가?
- numberBreaks (리스트) 의 요소와 letters (사전) 의 키가 항상 동일해야 하는데
 - 적어도, letters 로부터 numberBreaks 를 자동으로 만들어낼 수는 있을 것

코드 개선점 착안

- 순서쌍의 리스트 (list of tuples) 를 이용하면 보다 낫게 할 수 있는 않을까?

```
romans = [  
    (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')  
]
```

페이지를 넘기기 전에!

→ 직접 코드를 수정해 봅시다.

조금만 수정하면 됩니다.

로마 숫자 표현 함수 개선

```
romans = [  
    (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')  
]
```

```
result = "  
for value, letters in romans:  
    while n >= value:  
        result += letters  
        n -= value  
  
return result
```

리스트에 포함된 순서쌍들에 대하여
순서쌍 각 위치의 원소를
value 와 letter 에 담으면서
순환문을 반복

더 좋은 방법은 없을까?

리스트 대신 사전을 이용한다면?

```
romans = {  
    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'  
}
```

```
result = "  
for value in romans.keys():  
    while n >= value:  
        result += romans[value]  
        n -= value
```

```
return result
```

이 알고리즘이 올바르게 동작할까요?

아니라면, 왜 안될까요?

사전에 대한 반복

- 사전 d 가 있다고 할 때,
 - for x in d.keys(): 로 반복하면?
 - 사전의 원소들 사이에는 순서가 정해져 있지 않음
 - 키들 사이에 순서를 갖도록 순환시키려면?

```
result = "  
for value in sorted(romans.keys(), reverse=True):  
    while n >= value:  
        result += romans[value]  
        n -= value  
  
return result
```

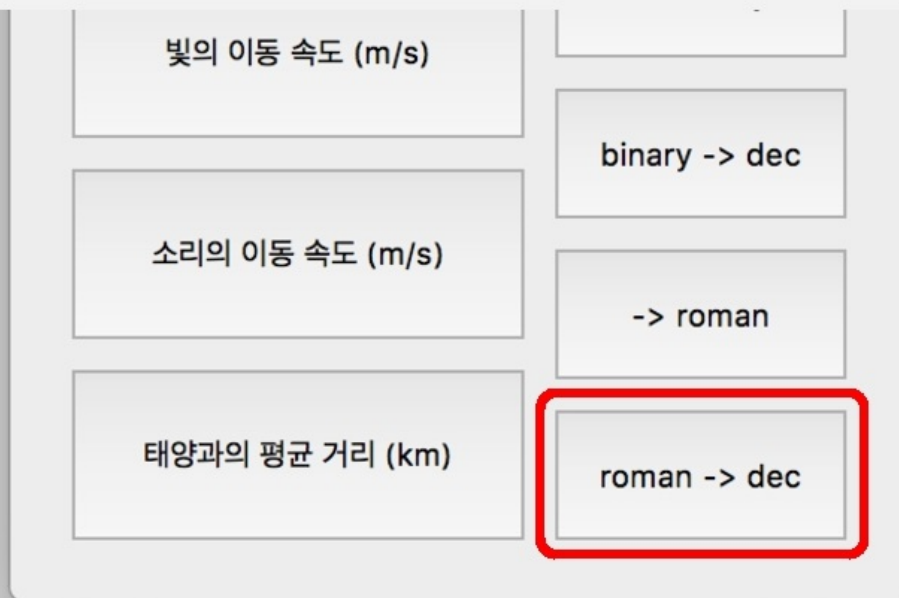
sorted (Python built-in function) 를 이용,
키에 대한 반복에 순서를 부여
(왜 reverse=True 가 필요한가요?)

이 방법이 아까의 방법에 비하여 장단점이 있나요?

과제: 로마숫자 → 10진수 변환 알고리즘 구현

- 기본 기능

- 계산 창에 로마숫자가 표시되어 있을 때 “roman → dec” 버튼을 누르면 해당 로마 숫자를 10진수로 변환하여 계산 창에 표시

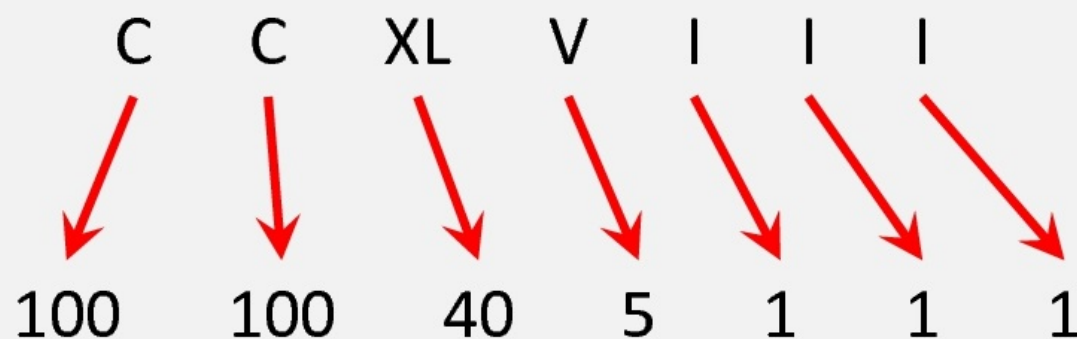


- 예외 처리

- 어떤 예외 상황이 발생할 수 있는지를 고려하여 예외 처리 코드를 추가

로마숫자 → 10진수: 알고리즘 착안점

248 = CCXLVIII

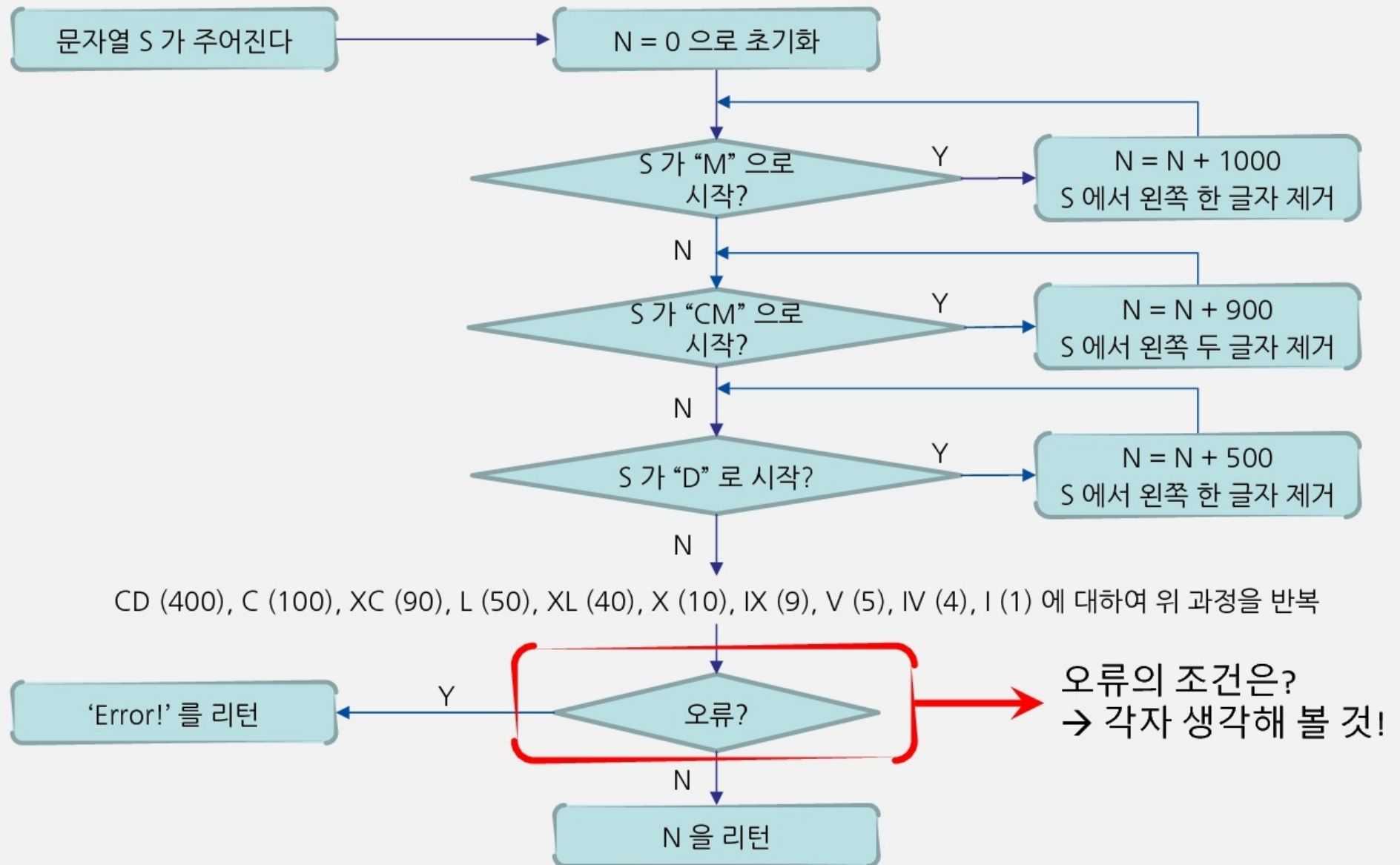


$$100 + 100 + 40 + 5 + 1 + 1 + 1 = 248$$

수	로마자
1	I
5	V
10	X
50	L
100	C
500	D
1000	M

1. 주어진 문자열은 왼쪽부터 차례로 살펴본다.
2. 큰 수에 해당하는 것부터 발견되는지 판단한다.
3. 발견되는 수는 임시 합에 더한다 (0 에서 시작).

로마숫자 → 10진수 알고리즘: 순서도



로마숫자 → 10진수 변환 알고리즘: 세부설계

- 문자열의 가장 왼쪽에 특정 문자열이 나타나는가를 판단
 - 문자열 슬라이싱 및 비교
 - 문자열에 대해서 find() 메서드를 이용
 - 예: 'CCXLVII'.find('C') == 0, 'CCXLVII'.find('XL') == 2
- 왼쪽부터 시작해서, 이미 처리한 부분 문자열은 그 다음부터는 고려하지 않는다.
 - 문자열 슬라이싱
- 예외처리
 - 만약 주어진 문자열이 올바른 로마 숫자 표현이 아니라면?
 - 이것을 어떻게 판단하는가?