Kata And Analysis

Jim Weirich
@jimweirich
Chief Scientist

Convert a decimal number to Roman Numerals

<< KATA >>

Know:

Where to start

```
describe RomanNumeralConverter do
  ??
end
```



```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumeralConverter.new }

  Then { converter.should_not be_nil }
end
```

```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumeralConverter.new }

  Then { converter.convert(1).should == “I” }
end
```


Know:

Where to continue

```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumberalConverter.new }

  Then { converter.convert(1).should == “I” }
end
```

Need to handle multiple “I”s

```
class RomanNumeralConverter
  def convert(number)
    "I"
  end
end
```



```
describe RomanNumeralConverter do
  Give(:converter) { RomanNumberalConverter.new }

  Then { converter.convert(1).should == “I” }
  Then { converter.convert(2).should == “II” }
end
```

Need to handle multiple “I”s

```
class RomanNumeralConverter
  def convert(number)
    "I" * number
  end
end
```

Know:

The solution to drive the tests


```
...
  Then { converter.convert(1).should == “I” }
  Then { converter.convert(2).should == “II” }
...
```

Need to handle “V”s

```
...
    "I" * number
...
```

```
...
  Then { converter.convert(1).should == “I” }
  Then { converter.convert(2).should == “II” }
  Then { converter.convert(5).should == “V” }
...
```

Need to handle “V”s

```
...
    if number == 5
      "V"
    else
      "I" * number
    end
...
```



```
...
  Then { converter.convert(5).should == “V” }
...
```

Need to get rid of the “ELSE”

```
...
    if number == 5
      "V"
    else
      "I" * number
    end
...
```

```
...
  Then { converter.convert(5).should == “V” }
  Then { converter.convert(5).should == “VI” }
...
```

Need to get rid of the “ELSE”

```
...
    result = ""

    if number >= 5
      result << "V"
      number -= 5
    end
    result << "I" * number

    result
...
```



```
...
  Then { converter.convert(5).should == “V” }
  Then { converter.convert(5).should == “VI” }
...
```

Need to get rid of the “ELSE”

```
...
    result = ""

    if number >= 5
      result << "V"
      number -= 5
    end
    result << "I" * number

    result
...
```

Forced:
*** Incremental Solution**

```
...
  Then { converter.convert(5).should == “V” }
  Then { converter.convert(5).should == “VI” }
...
```

Need to get rid of the “ELSE”

```
...
    result = ""

    if number >= 5
      result << "V"
      number -= 5
    end
    result << "I" * number

    result
...
```

Forced:
*** Incremental Solution**


```
...
  Then { converter.convert(5).should == “V” }
  Then { converter.convert(5).should == “VI” }
...
```

Need to get rid of the “ELSE”

```
...
    result = ""

    if number >= 5
      result << "V"
      number -= 5
    end
    result << "I" * number

    result
...
```

Forced:
*** Incremental Solution**

*** Reduction**

```
...
  Then { converter.convert(10).should == “X” }
...
```

Need to handle multiple “X”s

```
...
    if number >= 10
      result << "X"
      number -= 10
    end
...
```



```
...
  Then { converter.convert(10).should == “X” }
  Then { converter.convert(20).should == “XX” }
...
```

Need to handle multiple “X”s

```
...
    while number >= 10
      result << "X"
      number -= 10
    end
...
```

Know:

What to skip


```
...
  Then { converter.convert(1).should == “I” }
  Then { converter.convert(2).should == “II” }
  Then { converter.convert(4).should == “IV” }
  Then { converter.convert(5).should == “V” }
...
```

What about “IV”?

Know:

To recognize duplication


```
...
    while number >= 10
      result << "X"
      number -= 10
    end
    if number >= 9
      result << "IX"
      number -= 9
    end
    if number >= 5
      result << "V"
      number -= 5
    end
    if number >= 4
      result << "IV"
      number -= 4
    end
    result << "I" * number
...
```

- 3 Cases
  - While Loops
  - If Statements
  - * Statements

```
...
    while number >= 10
      result << "X"
      number -= 10
    end
    if number >= 9
      result << "IX"
      number -= 9
    end
    if number >= 5
      result << "V"
      number -= 5
    end
    if number >= 4
      result << "IV"
      number -= 4
    end
    while number >= 1
      result << "I"
      number -= 1
    end
...
```

- 2 Cases
  - While Loops
  - If Statements


```
...
    while number >= 10
      result << "X"
      number -= 10
    end
    while number >= 9
      result << "IX"
      number -= 9
    end
    while number >= 5
      result << "V"
      number -= 5
    end
    while number >= 4
      result << "IV"
      number -= 4
    end
    while number >= 1
      result << "I"
      number -= 1
    end
...
```

- 1 Case
  - While Loops

```
  CONVERSION_TABLE = [
    ["X", 10],
    ["IX", 9],
    ["V", 5],
    ["IV", 4],
    ["I", 1],
  ]
...
    result = ""
    CONVERSION_TABLE.each do |roman_digit, value|
      while number >= value
        result << roman_digit
        number -= value
      end
    end
    result
...
```



```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumeralConverter.new }

  Then { converter.convert(1).should == "I" }
  Then { converter.convert(2).should == "II" }
  Then { converter.convert(4).should == "IV" }
  Then { converter.convert(5).should == "V" }
  Then { converter.convert(6).should == "VI" }
  Then { converter.convert(9).should == "IX" }
  Then { converter.convert(10).should == "X" }
  Then { converter.convert(20).should == "XX" }
end
```

```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumeralConverter.new }

  def convert(number)
    converter.convert(number)
  end

  Then { convert(1).should == "I" }
  Then { convert(2).should == "II" }
  Then { convert(4).should == "IV" }
  Then { convert(5).should == "V" }
  Then { convert(6).should == "VI" }
  Then { convert(9).should == "IX" }
  Then { convert(10).should == "X" }
  Then { convert(20).should == "XX" }
end
```



```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumeralConverter.new }

  def convert(number)
    converter.convert(number)
  end

  Then { convert(1).should == "I" }
  Then { convert(2).should == "II" }
  Then { convert(4).should == "IV" }
  Then { convert(5).should == "V" }
  Then { convert(6).should == "VI" }
  Then { convert(9).should == "IX" }
  Then { convert(10).should == "X" }
  Then { convert(20).should == "XX" }
end
```

```
describe RomanNumeralConverter do
  Given(:converter) { RomanNumeralConverter.new }

  def convert(number)
    converter.to_roman(number)
  end

  Then { convert(1).should == "I" }
  Then { convert(2).should == "II" }
  Then { convert(4).should == "IV" }
  Then { convert(5).should == "V" }
  Then { convert(6).should == "VI" }
  Then { convert(9).should == "IX" }
  Then { convert(10).should == "X" }
  Then { convert(20).should == "XX" }
end
```


Know:

When to leave in duplication

```
CONVERSION_TABLE = [
  ["M", 1000],

  ["CM", 900],
  ["D", 500],
  ["CD", 400],
  ["C", 100],

  ["XC", 90],
  ["L", 50],
  ["XL", 40],
  ["X", 10],

  ["IX", 9],
  ["V", 5],
  ["IV", 4],
  ["I", 1],
]
```



```
def self.quad(power, ones, fives, tens)
  [
    ["#{ones}#{tens}", 9*power],
    ["#{fives}", 5*power],
    ["#{ones}#{fives}", 4*power],
    ["#{ones}", 1*power],
  ]
end

CONVERSION_TABLE = [
  ["M", 1000],
] +
  quad(100, "C", "D", "M") +
  quad(10,  "X", "L", "C") +
  quad(1,   "I", "V", "X")
```

```
Then { convert(1).should    == "I" }
Then { convert(2).should    == "II" }
Then { convert(4).should    == "IV" }
Then { convert(5).should    == "V" }
Then { convert(6).should    == "VI" }
Then { convert(9).should    == "IX" }
Then { convert(10).should   == "X" }
Then { convert(20).should   == "XX" }
Then { convert(40).should   == "XL" }
Then { convert(50).should   == "L" }
Then { convert(90).should   == "XC" }
Then { convert(100).should  == "C" }
Then { convert(400).should  == "CD" }
Then { convert(500).should  == "D" }
Then { convert(900).should  == "CM" }
Then { convert(1000).should == "M" }
```



```
[
  [1, "I"],
  [2, "II"],
  [4, "IV"],
  [5, "V"],
  [6, "VI"],
  [9, "IX"],
  [10, "X"],
  [20, "XX"],
  [40, "XL"],
  [50, "L"],
  [90, "XC"],
  [100, "C"],
  [400, "CD"],
  [500, "D"],
  [900, "CM"],
  [1000, "M"],
].each do |value, roman|
  convert(value).should == roman
end
```

Know:

The edgecases


```
Then { convert(2013).should == "MMXIII" }
Then { convert(1949).should == "MCMXLIX" }
Then { convert(3999).should == "MMMCMXCIX" }
Then { convert(0).should    == "" }
```

```
Then { convert(2013).should == "MMXIII" }
Then { convert(1949).should == "MCMXLIX" }
Then { convert(3999).should == "MMMCMXCIX" }
Then { convert(0).should    == "" }
```



```
Then { convert(2013).should == "MMXIII" }
Then { convert(1949).should == "MCMXLIX" }
Then { convert(3999).should == "MMMCMXCIX" }
Then { convert(0).should    == "" }
```

```
Then { convert(2013).should == "MMXIII" }
Then { convert(1949).should == "MCMXLIX" }
Then { convert(3999).should == "MMMCMXCIX" }
Then { convert(0).should    == "" }
```



```
Then { convert(2013).should == "MMXIII" }
Then { convert(1949).should == "MCMXLIX" }
Then { convert(3999).should == "MMMCMXCIX" }
Then { convert(0).should    == "" }
```

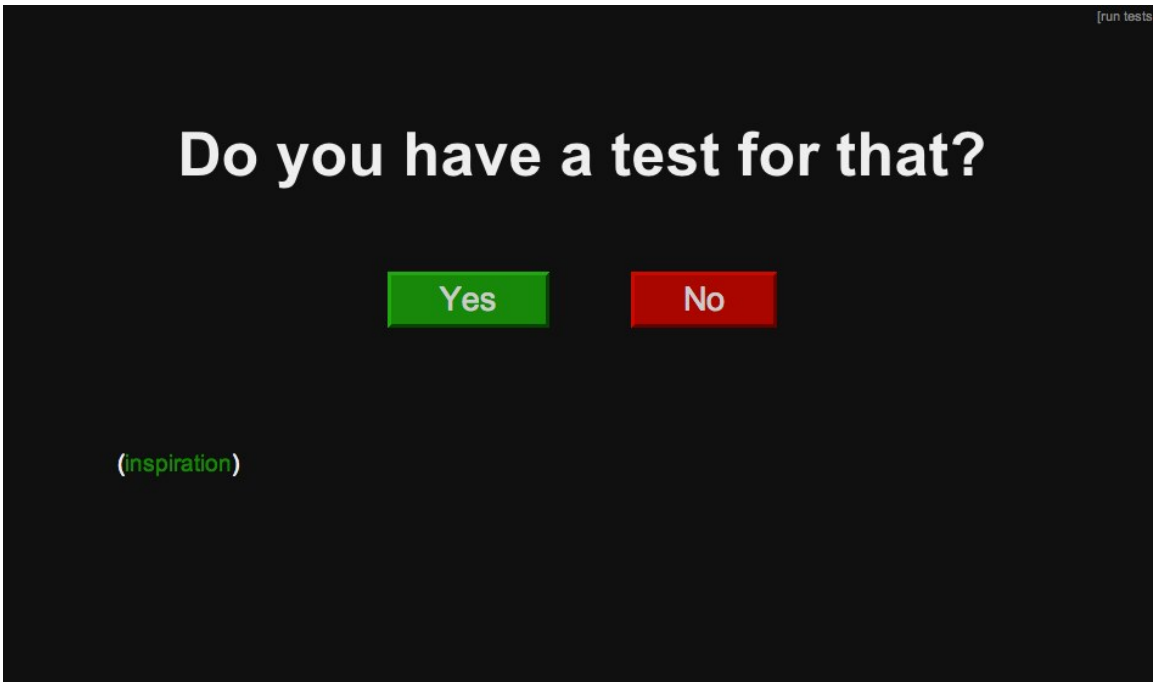
Know ...

- Where to start
- Where to continue
- To let the solution drive the tests
- What to skip
- To recognize duplication
- When to leave duplication
- The edgecases

Resources

- Code Katas: http://codekata.pragprog.com/
- RNC Description: http://codingdojo.org/cgi-bin/wiki.pl?KataRomanCalculator
- Roman Numeral Live Performances
  - Enrique Comba Riepenhausen: http://katas.softwarecraftsmanship.org/?p=21
  - Javier Acero: http://vimeo.com/20765638
- Rspec-Given: http://github.com/jimweirich/rspec-given

http://pairprogrammingbot.com



Resources

http://github.com/jimweirich/pair_programming_bot

http://RubyMotion.com

Questions?

Thank You

Jim Weirich
Chief Scientist

neo

jim@neo.com
@jimweirich