Roman DSL

1. Goals

- Support roman numbers within the host-language: **`III + IV = VII`**
- Keep syntax of host-language.
- Avoid boilerplate of any kind.

2. Implementation

- Show method **`RomanDSL class>>#romanToArabic:`**.
- Implement the transformation on the class side:

```
RomanDSL class>>transformRoman
    <transform>

    ^ DSLTreePattern new
        expression: '`var' do: [ :context |
            | arabic |
            arabic := self romanToArabic: context node name.
            arabic notNil
                ifTrue: [ context node swapWith: arabic lift ] ]
```

- **`<transform>`** tells the compiler that this is a transformation rule.
- **`DSLTreePattern`** defines the scope **`` `var``** of an action to be performed on the parse tree.
- The action block calls **`#romanToArabic:`** to transform the roman number to an **`Integer`** object.

- If the node is actually a roman number (**`#notNil`**), replace the it (**`#swapWith:`**) with the arabic number.
- **`#lift`** turns the **`Integer`** into a **`LiteralNode`**.

3. Test it

- Implement a test-case on the instance side:

```
RomanDSL>>testAdd
    self assert: III + IV = VII
```

- The test passes (⌘**`T`**).
- Show decompiled code (View | Decompile).

4. Debug it

- Put a **`#halt`** at the beginning of **`RomanDSL>>testAdd`**.
- Run the test and step through it.

5. Other Examples

- **`CUBrainfuckExample`**
  Completely new language within the host-language.
  Evaluate **`CUBrainfuckExample debug`** to show debugger.
- **`CUSwapExample`** / **`CUControlExample`**
  Add new features to host-language.