Mini-DSL

1. Goal

- 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>>transformation
    <transform>

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

- **`<transform: 100>`** 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 (**`#replaceWith:`**) 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.
- Show decompiled code.

4. Other Examples

- **`CUInterpolateExample`**
  Similar, but more realistic use-case.
- **`CUSwapExample`** / **`CUCastingExample`**
  Add new features to host-language.
- **`FLFactorialExample`**
  Completely new language within the host-language.