Seven More Languages in Seven Weeks

Correl Roush

January 27, 2016

FACTOR

A stack-based, concatenative programming language

- Installing Factor
- Using the REPL
- Basic Syntax & Data Types
- Stack Shuffling
- Combinators

```
"Hello, world" print
! Hello, world

"same" length "diff" length = .
! t
```

Booleans `t` or `f`

Sequences

Lists `{ 4 3 2 1 }`

Maps `{ { "one" 1 } { "two" 2 } { "three" 3 } }`

Quotations `[ 42 + ]`

Conditionals take quotations as branching arguments

```
10 0 > [ "pos" ] [ "neg" ] if .
! pos
-5 0 > [ "pos" ] [ "neg" ] if .
! neg
```

dup	Duplicate a value on the stack
drop	Drop the top value from the stack
nip	Drop the second value
swap	Swap two values
over	Duplicates the second value over to the top
rot	Rotate the top 3 values on the stack

- bi, bi@, bi*
- tri, tri@, tri*
- dip, keep

```
44.50 [ 0.05 * ] [ 0.09975 * ] bi
! 2.225
! 4.438875

44.50 22.50 [ 0.05 * ] bi@
! 2.225
! 1.125

44.50 22.50 [ 0.05 * ] [ 0.09975 * ] bi*
! 2.225
! 2.244375
```

Exercises

- Defining Words
- Vocabularies
- Unit Tests
- Interview with Slava Pestov

```
: add-42 ( x -- y ) 42 + ;

: sum ( seq -- n ) 0 [ + ] reduce ;

: first-two ( seq -- a b ) [ first ] [ second ] bi ;
```

Words are organized into vocabularies, which are similar to packages, modules, or namespaces in other languages.

Factor includes a unit testing vocabulary (`tools.test`), which is useful for ensuring correctness of your code, and also experimenting with the language.

```
USING: examples.greeter tools.test ;
IN: examples.greeter.tests

{ "Hello, Test" } [ "Test" greeting ] unit-test
```

I decided to write my own language, though, because I wanted something really simple, and also just because it would be fun.

Exercises

- Tuples
- Pipelining with Higher-Order Words

01 Defining

```
TUPLE: name slot ... ;
```

02 Accessing and Modifying

- `slot>>`
- `>>slot`
- `change-slot`

03 Creating

- `boa` (*By Order of Arguments*)
- `T{ name { slot value } ... }`

```
CONSTANT: gst-rate 0.05
CONSTANT: pst-rate 0.09975

: gst-pst ( price -- taxes ) [ gst-rate * ] [ pst-rate * ] bi + ;

: taxes ( checkout taxes-calc -- taxes )
    [ dup base-price>> ] dip
    call >>taxes ; inline
```

The `inline` keyword is necessary, as the `taxes` word takes quotations as parameters.

Strengths

- Simple syntax
- Easy function composition
- Batteries included

Weaknesses

- Learning curve
- Small community
- Limited resources

Final Thoughts