# Seven Languages in Seven Weeks

Correl Roush

June 10, 2015

| | |
|---|---|
| Created | 2002 |
| Author | Steve Dekorte |

An embeddable prototype language with a simple syntax and strong concurrency model.

## GETTING IO

http://iolanguage.org/

## DAY 1: SKIPPING SCHOOL, HANGING OUT

- Prototypes
- Sending messages
- Objects, slots, and types
- Methods
- Collections
- Singletons

## OBJECTS, PROTOTYPES, AND INHERITANCE

```
Vehicle := Object clone
Vehicle description := "Something to take you far away"
Car := Vehicle clone
Car description // "Something to take you far away"
ferrari := Car clone
```

- Objects are just containers of slots. If the slot isn't there, Io calls the parent.
- `ferrari` has no type slot. By convention, types in Io begin with uppercase letters.

## METHODS

```
Car drive := method("Vroom" println)
```

- A method is an object.
- Since a method is an object, we can assign it to a slot.
- If a slot has a method, invoking the slot invokes the method.

## LISTS AND MAPS

```
list(1, 2, 3, 4) sum // 10

elvis := Map clone
elvis atPut("home", "Graceland")
elvis at("home") // "Graceland"
```

- A list is an ordered collection of objects of any type.
- A hash is a lot like an Io object in structure where the keys are slots that are tied to values.

## TRUE, FALSE, NIL, AND SINGLETONS

```
4 < 5           // true
4 <= 3          // false
true and false  // false
true or false   // true
true and 6      // true
true and 0      // true

true clone      // true
false clone     // false
nil clone       // nil
```

`true`, `false`, and `nil` are singletons.

```
Highlander := Object clone
Highlander clone := Highlander
```

We've simply redefined the `clone` method to return `Highlander`, rather than letting Io forward requests up the tree, eventually getting to `Object`.

## EXERCISES

## DAY 2: THE SAUSAGE KING

- Loops & Conditionals
- Operators
- Messages
- Reflection

## LOOPS

### Infinite loop

```
loop("getting dizzy..." println)
```

### While loop

```
i := 1
while(i <= 11, i println; i = i + 1); "This one goes up to 11" println
```

### For loop

```
for(i, 1, 11, i println); "This one goes up to 11" println
```

## CONDITIONALS

The `if` control structure is implemented as a function with the form `if(condition, true code, false code)`. The function will execute `true code` if `condition` is `true`; otherwise, it will execute `false code`.

## OPERATORS

| | |
|---|---|
| 0 | `? @ @@` |
| 1 | `**` |
| 2 | `% * /` |
| 3 | `+ -` |
| 4 | `<< >>` |
| 5 | `< <= > >=` |
| 6 | `!= ==` |
| 7 | `&` |
| 8 | `^` |
| 9 | `\vert` |
| 10 | `&& and` |
| 11 | `or` |
| 12 | `..` |
| 13 | `%= &= *= += -= /= <<= >>= ^= \vert=` |
| 14 | `return` |

## ASSIGN OPERATORS

```
::=  newSlot
:=   setSlot
=    updateSlot
```

## ADD AN OPERATOR

```
OperatorTable addOperator("xor", 11)

true xor := method(bool, if(bool, false, true))
false xor := method(bool, if(bool, true, false))
```

## MESSAGES

A message has three components: the `sender`, the `target`, and the `arguments`.
The `call` method gives you access to the meta information about any message.

```
Object ancestors := method(
        prototype := self proto
        if(prototype != Object,
        writeln("Slots of ", prototype type, "\n---------------")
        prototype slotNames foreach(slotName, writeln(slotName))
        writeln
        prototype ancestors))


Animal := Object clone
Animal speak := method(
            "ambiguous animal noise" println)

Duck := Animal clone
Duck speak := method(
            "quack" println)

Duck walk := method(
            "waddle" println)

disco := Duck clone
disco ancestors
```

EXERCISES

## DAY 3: THE PARADE AND OTHER STRANGE PLACES

- DSLs
- Metaprogramming
- Concurrency

```
OperatorTable addAssignOperator(":", "atPutNumber")
curlyBrackets := method(
  r := Map clone
  call message arguments foreach(arg,
       r doMessage(arg)
       )
  r
)
Map atPutNumber := method(
  self atPut(
       call evalArgAt(0) asMutable removePrefix("\"") removeSuffix("\"")
       call evalArgAt(1))
)
s := File with("phonebook.txt") openForReading contents
phoneNumbers := doString(s)
phoneNumbers keys   println
phoneNumbers values println
```

## IO'S METHOD_MISSING

```
Builder := Object clone
Builder forward := method(
  writeln("<", call message name, ">")
  call message arguments foreach(
        arg,
        content := self doMessage(arg);
        if(content type == "Sequence", writeln(content)))
  writeln("</", call message name, ">"))
Builder  ul(
        li("Io"),
        li("Lua"),
        li("JavaScript"))
```

## CONCURRENCY

- Coroutines
- Actors
- Futures

## EXERCISES

## WRAPPING UP: STRENGTHS

- Footprint
- Simplicity
- Flexibility
- Concurrency

## WRAPPING UP: WEAKNESSES

- Syntax
- Community
- Performance

## FINAL THOUGHTS

*Like Lisp, Io has a strong overriding philosophy of simplicity and flexibility.*