1 Scam vs Scheme

Things to note for programmers who are trying Scam:

variable names

Scam variable/function names are case-sensitive.

assignment

The Scam version of set! has an additional mode, when compared to Scheme; the function takes an optional environment, where the predefined variable this always points to the current environment. Thus, the following two expressions are equivalent:

(\texttt{(set! x 5)})
(\texttt{(set! x 5 this)})

There is a version of set! that evaluates all its arguments; to assign to a variable, one quotes the variable name:

(\texttt{(set 'x 5)})

The set function also takes an environment as an optional third argument.

variadic functions

If the last formal parameter in a function definition is the symbol @, all remaining arguments not matched with preceding formal parameters are bundled up into a list, with the variable @set to point to that list. Here is a redefinition of the built-in function println that has the same semantics:
(define (println @)
  (while (valid? @)
    (print (car @))
    (set! @ (cdr @))
  )
  (print "\n")
)

special forms

There are no special forms in Scam; all functions can be redefined, including if, and, or, and define.

objects

Class and constructor are the same thing in Scam. Any function that returns the pre-defined variable this is considered a class definition and a constructor for that class. Here is an example Node class:

(define (Node value next)
  this
)

There are two methods for extracting object components, dot and get. The latter two expressions have identical semantics:

(define n (Node 3 nil))
(dot n value)
(get 'value n)

The dot function does not evaluate its last argument. Like set, get evaluates all its arguments.

Note that environments and objects are equivalent in Scam.

object methods

Nested functions in a constructor are methods:

(define (Node value next)
  (define (toString)
    (+ "Node(" (string value) "," (string next) ")")
  )
)

2
Calling object methods proceeds as expected:

```
(define n (Node 3 nil))
((get 'toString n))
```

**inheritance**

Inheritance is not native to Scam, but is accomplished by including the inheritance library:

```
(include "inherit.lib")
```

Once included, the *new* function is used to perform inheritance:

```
(define (A)
  (define parent nil)
  this)

(define (B)
  (define parent (A))
  this)

(define obj (new (B)))
```

Note that constructors in an inheritance hierarchy must, by convention, define a *parent* variable.

Inheritance is similar to Java; every method is virtual. Unlike Java, ancestor objects ‘inherit’ the enclosing scope of the child object. In other words, if an ancestor method references a non-local variable, the non-local is resolved in the scope of the child object and, if not resolved there, in the child’s enclosing scope.