

Class 18: Iterating Over Lists

Held: Wednesday, 20 February 2008

Summary: We consider procedures for that involve doing something with each element of a list. Along the way, we consider two key ideas of functional programming: *higher-order programming* and *anonymous procedures*.

Related Pages:

- EBoard.
- Lab: Iterating Over Lists.
- Reading: Iterating Over Lists.
- Due: Assignment 4: Blending Colors.

Notes:

- The DrFu Web site now has links to instructions for running DrFu remotely on your PC or Linux box. Let me know if they work okay.
- I think everyone got their homework in by midnight (well 12:05 a.m.) last night. Thanks!
- Ian Athanasakis will present a talk on his work at Google on Thursday at 4:30 in 3821. Extra credit for attending.
- Friday's reading on local bindings will not be available until tonight or tomorrow.
- I'll reserve a few minutes at the start of class to consider conditionals..

Overview:

- Review: How Scheme evaluates expressions.
- Repetition.
- Building new lists from old with `map`.
- Anonymous procedures.
- Doing something with each value in a list with `foreach!`.
- Drawing lists of spots.

Review: How Scheme evaluates expressions

Let's consider the steps involved in evaluating

```
(define x 5)
(define y 3)
(+ x y)
```

Now, let's try the following somewhat more complicated example.

```
(define x 5)
(define y 7)
(define fun (lambda (x) (* x y)))
(fun 3)
```

Repetition

- Key algorithm design ideas:
 - *Naming*: It is useful to name values
 - *Sequencing*: Algorithms often involve executing operations in a defined sequence
 - *Conditionals*: We often need to choose between actions
 - *Procedures*: We clarify and simplify our code by parameterizing and grouping collections of instructions.
 - *Repetition*: Algorithms often have to perform activities repeatedly (for a fixed number of repetitions, until some condition holds, or, potentially “forever”).
- In today's class, we'll consider the special case of repetition for lists.
- Two approaches:
 - *Pure*: Create a new list from the old list.
 - *Impure*: Do some separate activity using each value.
- A pure approach: Build a new list of spots by shifting each spot right 4 spaces.
- An impure approach: Render each spot in a list of spots.
- The procedure for the first approach is `map`. The procedure for the second approach is `foreach!`.

Building new lists from old with `map`

- Form: `(map function (list v1 v2 ... vn))`
- Meaning: `(list (function v1) (function v2) (function v3) ... (function vn))`
- Examples ...

Anonymous procedures

Let's return to the examples. What happens if we write

```
(define triple (lambda (x) (* x 3)))
(map triple (list 1 2 3))
```

As we'll see, the interpreter substitutes the lambda expression for the `triple`. We can do the same.

```
(map (lambda (x) (* x 3)) (list 1 2 3))
```

Doing something with each value in a list with `foreach!`

- Form: `(foreach! function (list v1 v2 vn))`
- Meaning:

```
(function v1)
(function v2)
...
(function vn)
null
```

Copyright © 2007-8 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.