

6.001 Tutorial 13

David Ziegler

December 6, 2004

1 Administrivia

- The last tutorial!
- Last lecture is on Thursday, 10:00, 32-123!
- Project 5 will be back at the final.
- No office hours on Tuesday; finals week office hours to be arranged.
- Problem set 10 (feedback) – please do it, we really do care!
- **Important** — I’m teaching recitation on Wednesday!

2 Lazy Evaluation

The whole point of lazy evaluation is to put off doing work as long as you can. There are two ways of doing that. One is to be explicit:

```
(delay exp)
; Returns a promise to evaluate exp
; in current environment

(force promise)
; Returns the value of exp in the
; original environment
```

It’s easier if everything is implicit. In this case, all arguments to functions are delayed, and values are forced when needed:

- Printing the value to screen.
- Used as an argument to a primitive procedure.
- Used in a conditional.
- Used as an operator.

Lazy evaluation has a bunch of interesting applications, but the one that we’re most interested in is constructing streams, (usually) infinite data structures. A stream is just like a list, but the `cdr` of each cons cell is lazy-memoized.

```
(define (cons-stream x (y lazy-memo))
  (cons x y))
```

```
(define stream-car car)
```

```
(define stream-cdr cdr)
```

Some streams are very easy to construct. For example, the stream of infinite 1’s:

```
(define ones (cons-stream 1 ones))
```

For more complicated streams, we need some helper functions:

```
(define (add-streams s1 s2)
```

To construct a stream, think about the mathematical relationship between successive elements in the stream. For example, we know that the n th Fibonacci number is the sum of the $n - 1$ st and $n - 2$ nd Fibonacci numbers, and that the first and second Fibonacci numbers are both 1.

```
(define fibs
  (cons-stream 1
    (cons-stream 1
```

In general, to figure out how to write out code to generate a stream, write out the numbers in the stream, and then figure out a relationship between them. Figure out how to express some element of the stream in terms of other elements that precede.

```
(define facts
```

3 Asynchronous Computing

The whole idea of asynchronous computing is that two processes that have shared state might interact in different ways if they are running in parallel.

```
(define x 0)
```

```
(define (foo)
  (set! x 1))
```

```
(define (bar)
  (set! x 2))
```

```
(parallel-execute foo bar)
```

So why do we care?

4 Review

Topics from this course:

- Scheme
- Procedures and recursion
- Orders of growth
- Data abstractions
- Higher-order procedures
- Program methodology
- Symbols and quotation
- Abstract data types
- Mutation
- Environment model
- Object-oriented systems
- Interpretation/evaluation
- Lazy evaluation
- Asynchronous computing

Final Feedback

1. How have you enjoyed/tolerated/hated 6.001?
2. What things have gone well?
3. What things have not?
4. What did you think of tutorial? Was it useful?
5. Are there any things you would have liked to have seen from me that would have helped?
6. If you could change anything about tutorial, what would it be?
7. Any other comments?

Thanks for your feedback!