Lecture 2: More on Functional Programming

CS 61A
Summer 2006

Administrative stuff

- Check the website often!
- Activate your cardkeys at 387 Soda
- Read the course info handout, the Unix handout, the Emacs handout (all are online) if you haven’t already.
- Go to lab today, not discussion!!!
- HW1A posted, Project 1 posted
- Readings for this week: 1.1, 1.3

Functions vs. Procedures

- Consider:
  - f(x) = 3x+6
  - g(x) = 3(x+2)
- Same function?
- Same procedure?
- Most of the time we use the word “function” to mean “procedure”; this won’t cause any confusion.

Review

Terminology:
We write Scheme EXPRESSIONS; Scheme (or STk) EVALUATES the expressions and RETURNS the VALUE of the expression.

Review

> 3
  3
> (+ 1 2 (* 3 4))
  15
> (+)
  0
> (*)
  1

Review

> +
  #[closure ...]
> hello
  *** Error: unbound variable: hello
> (/ 1 0)
  *** Error: divide by zero
Review

A word is a quoted string of non-space characters (numbers, letters, punctuation).
> 'hello
hello
> (first (butfirst (butfirst 'abcde)))
<
> (word (bl 'kobe) (bf 'jordan))
kobordan
> (word 'z? 'some '+ 'stuff '+)
z?some+stuff!

Review

A sentence is a flat sequence of words.
Sentences can also be quoted.
> (sentence 'the 'cat 'in 'the 'hat)
(the cat in the hat)
> (bl '(the cat in the hat))
(the cat in the)
> (se '(the cat) '(in the hat))
(the cat in the hat)

Review

Some more examples:
> (se (+ 1 (word '1 1)) (se 'burrito))
(12 burrito)
> (bf '(foo))
()
> (bl 'f)

""
> (se '() '(the ocean))
(the ocean)

Review

Some more examples:
> (se #t 'this is a sentence))
(12 burrito)
> (bf '(foo))
()
> (bl 'f)

""
> (se '() '(the ocean))
(the ocean)

Review

Some more examples:
> (empty? '(())
#t
> (empty? "")
#t
> (equal? 'the cat) '(the dog))
#f
> (member? 40 '(abba zabba 40 zing))
#t

Review

#t and #f are special symbols (booleans) that we use to denote true or false. They are not considered words.
> (se #t 'this is a sentence))
(12 burrito)
> (bf '(foo))
()
> (bl 'f)

""
> (se '() '(the ocean))
(the ocean)

In Scheme, when we make a PROCEDURE call, all arguments are ALWAYS evaluated. Special forms follow their own special rules and do not necessarily have to evaluate all their arguments nor follow standard rules of syntax. `if` is an example of a special form.

### Defining variables:

```scheme
> (define my-fav-number 139372929)
my-fav-number
> my-fav-number
139372929
> (+ my-fav-number 1)
139372930
> (first my-fav-number)
1
```

### Defining procedures:

```scheme
(define (my-func x y)
  (+ x y (* x y)))
x and y are the formal parameters
“(+ x y (* x y))” is the body of the procedure
(my-func 2 2)
=> (+ x y (* x y))
=> (+ 2 2 (* 2 2))
=> (+ 2 2 4)
8
Remember, define is a special form!!!
```

### Applicative order evaluation

```scheme
(define (f x y) (+ x y 1))
(define (g a) (f (+ a 1) (+ a 2)))
(g (+ 2 1))
=> (g 2) [eval the arguments]
=> (+ a 1) (+ a 2) [body of the proc]
=> (+ 2 1) (+ 2 2) [replace a with its value 2]
=> (+ 3 4) [eval the arguments]
=> (+ x y 1) [body of the proc]
=> (+ 3 4 1) [replace parameters with values]
=> 8 [final result]
```

### Normal order evaluation (“lazy”)

```scheme
(define (square x) (* x x))
(square (square 2))
```

Applicative order is sometimes more efficient than normal order evaluation. Consider:

```scheme
(define (square x) (* x x))
(square (square 2))
```

Sometimes the two can give different results. Consider:

```scheme
(define (f x y) (+ x y))
(f 22 (/ 1 0))
```
Recursion

(define (pigl wd)
  (if (pl-done? wd)
      (word wd 'ay)
      (pigl (word (bf wd) (first wd)))))))

(define (pl-done? wd))

(define (vowel? letter)
  (member? letter '(a e i o u)))

What would happen if we tried to do (pigl 'fly)?

Recursion

Let's play with STk!
Factorial example
Argue example
Pigl sentence example
General example of every

Recursion

Computing factorials:
If N=1, then N! = 1
If N>1, then
N! = N*(N-1)*(N-2)*...*3*2*1
= N*(N-1)!)

(define (fact n)
  (if (= n 1) "; base case"
    1
    (* n (fact (- n 1))) )) ; recursive case

Recursion

(define (opposite w)
  (cond 
    ((equal? w 'great) 'terrible)
    ((equal? w 'terrible) 'great)
    ((equal? w 'good) 'bad)
    ((equal? w 'bad) 'good)
    ((equal? w 'like) 'dislike)
    ((equal? w 'dislike) 'like)
    ((equal? w 'love) 'hate)
    ((equal? w 'hate) 'love)
    (else w)))

Recursion

(define (argue sent)
  (if (empty? sent)
      '()
      (se (opposite (first sent))
          (argue (bf sent)))))

(define (pigl-sent sent)
  (if (empty? sent)
      '()
      (se (pigl (first sent))
          (pigl-sent (bf sent)))))

Recursion

(define (every proc sent)
  (if (empty? sent)
      '()
      (se (proc (first sent))
          (every proc (bf sent)))))