CS61BL SPRING 2011

COLLEEN LEWIS

The Interpreter + Calling Functions

```
STk> 3
3

STk>(+ 3 4)
7

STk> (+ 3 4 5 6)
18
```

Why not 3 + 4

Here we were calling the function +

The Interpreter + Calling Functions

```
STk> (sqrt 16)
4

STk>(+ 3 (sqrt 16))
7

STk> (+ 3 4 5 6)
18
```

Not all procedures are punctuation

Calling two functions Work from the inside-out

Scheme was designed by people

```
STk> (+)
0

STk>(*)
1

STk> (- 9 5 2)
2
```

Someone designed these to make sense. EVERYTHING should make sense!

You might disagree, but you should still understand the rationale

Parentheses Matter

```
STk> 9
9

STk> +
#[closure arglist=args 196d20]

STk> *
#[closure arglist=args 1970d0]
```

We asked scheme what this is, it said: 9

We asked scheme what this is, it said: a procedure

BIG IDEA: Procedures are a “thing”

You can not include extra parentheses!

Clickers

- Clickers are required (register online)
- Helps **me** see what concepts are challenging for the class
- Helps **you** see what concepts are challenging for you
- Provides
  - Explanations from peers
  - Experience explaining tough concepts

TRY IT

```
STk>(+ 7(* 3 4)(* (/ 10 5)(- 3 10
```

How many parentheses do we need at the end?

```
a) 0
b) 1
c) 2
d) 3
e) 4
```

If you finish early calculate the answer!

Quoting stuff

```
STk> '+
+
STk> 'hello
hello
STk> '(+ 2 3)
(+ 2 3)
STk>'(good morning))
(good morning)
```

These are `words`

These are `sentences`

Notice the lack of quote on the result

Functions for `words/sentences`

```
STk> (first 274)
2
STk> (butfirst 274)
74
STk> (first 'hello)
h
STk> (first '(hello))
hello
```

This is a word

This is a sentence

Sort of weird that they work on: `numbers` `words` `sentences`

REVIEW: Two Types of ('s so far

```
(first '(hello))
```

Call a function

Indicate a sentence

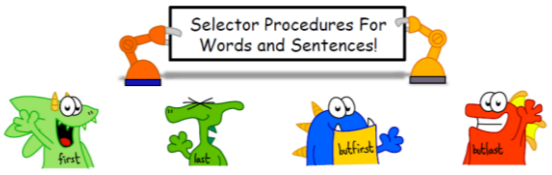
REVIEW: Two Types of ''s so far

```
'hi        '(hello)
```

This is a word

This is a sentence

http://csillustrated.berkeley.edu



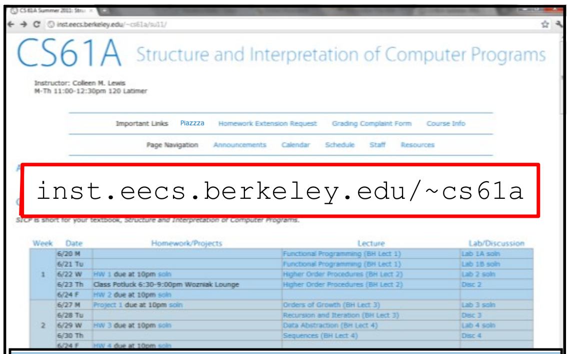
Functions for words/sentences

```
STk> (first (butfirst 'hello))
e

STk> (se (first 23) (last 45))
24
```

Work from the inside out.
10% knowledge
90% care

TRY IT

```
STk> (first '(hi))
hi
STk> (butfirst '(hi))
a)  i
b)  'i
c)           <= this is blank
d)  ()
e)  '()
```



CS61A Structure and Interpretation of Computer Programs

Instructor: Colleen M. Lewis
M-Th 11:00-12:30pm 120 Latimer

Important Links Piazzza Homework Extension Request Grading Complaint Form Course Info

Page Navigation Announcements Calendar Schedule Staff Resources

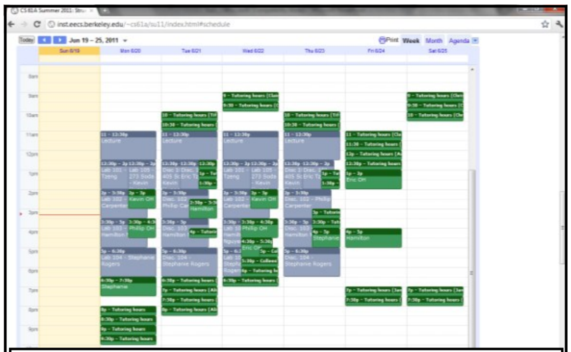
inst.eecs.berkeley.edu/~cs61a

SICP is short for your textbook, Structure and Interpretation of Computer Programs.

Week	Date	Homework/Projects	Lecture	Lab/Discussion
1	6/20 M		Functional Programming (BH Lect 1)	Lab 1A soln
	6/21 Tu		Functional Programming (BH Lect 1)	Lab 1B soln
	6/22 W	HW 1 due at 10pm soln	Higher Order Procedures (BH Lect 2)	Lab 2 soln
	6/23 Th	Class Potluck 6:30-9:00pm Wozniak Lounge	Higher Order Procedures (BH Lect 2)	Disc 2
	6/24 F	HW 2 due at 10pm soln		
2	6/27 M	Project 1 due at 10pm soln	Orders of Growth (BH Lect 3)	Lab 3 soln
	6/28 Tu		Recursion and Iteration (BH Lect 3)	Disc 3
	6/29 W	HW 3 due at 10pm soln	Data Abstraction (BH Lect 4)	Lab 4 soln
	6/30 Th		Sequences (BH Lect 4)	Disc 4
	6/24 F	HW 4 due at 10pm soln		

Important Links Piazzza Homework Extension Request Grading Complaint Form Course Info

Page Navigation Announcements Calendar Schedule Staff Resources



Jun 19 – 25, 2011

Important Links Piazzza Homework Extension Request Grading Complaint Form Course Info

Page Navigation Announcements Calendar Schedule Staff Resources

Staff

Instructor: Colleen Lewis
colleenL@berkeley.edu

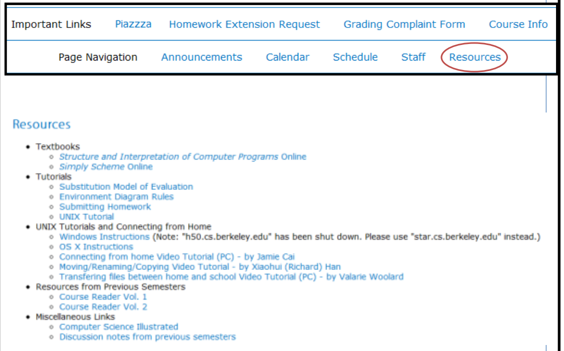
TA: Eric Tseng
12:30-2:00 (101)
cs61a-te@imail.eecs.berkeley

TA: Kevin Hwang
12:30-2:00 (105)
cs61a-td@imail.eecs.berkeley

TA: Phillip Carpenter
2:00-3:30 (102)
cs61a-tc@imail.eecs.berkeley

TA: Hamilton Nguyen
3:30-5:00 (103)
cs61a-tb@imail.eecs.berkeley

TA: Stephanie Rogers
5:00-6:30 (104)
cs61a-ta@imail.eecs.berkeley

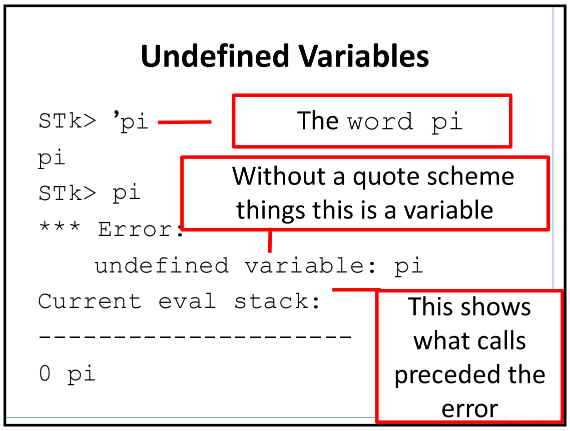


Important Links Piazzza Homework Extension Request Grading Complaint Form Course Info

Page Navigation Announcements Calendar Schedule Staff Resources

Resources

- Textbooks
  - *Structure and Interpretation of Computer Programs* Online
  - *Simply Scheme* Online
- Tutorials
  - Substitution Model of Evaluation
  - Environment Diagram Rules
  - Submitting Homework
  - UNIX Tutorial
- UNIX Tutorials and Connecting from Home
  - Windows Instructions (Note: "h50.cs.berkeley.edu" has been shut down. Please use "star.cs.berkeley.edu" instead.)
  - OS X Instructions
  - Connecting from home Video Tutorial (PC) - by Jamie Cai
  - Moving/Renaming/Copying Video Tutorial - by Xiaohui (Richard) Han
  - Transfering files between home and school Video Tutorial (PC) - by Valarie Woolard
- Resources from Previous Semesters
  - Course Reader Vol. 1
  - Course Reader Vol. 2
- Miscellaneous Links
  - Computer Science Illustrated
  - Discussion notes from previous semesters



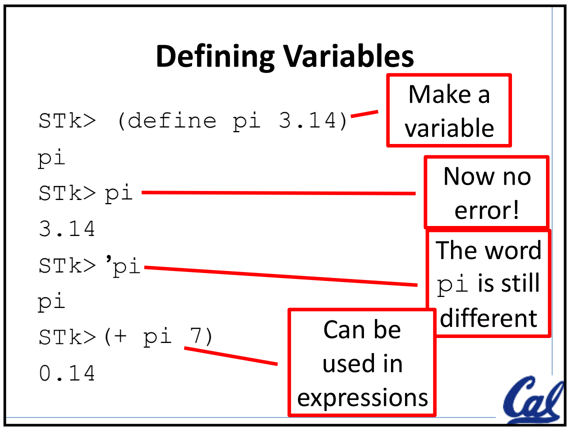
Undefined Variables

```
STk> 'pi
pi
STk> pi
*** Error:
    undefined variable: pi
Current eval stack:
---------------------
0 pi
```

The word pi

Without a quote scheme things this is a variable

This shows what calls preceded the error



Defining Variables

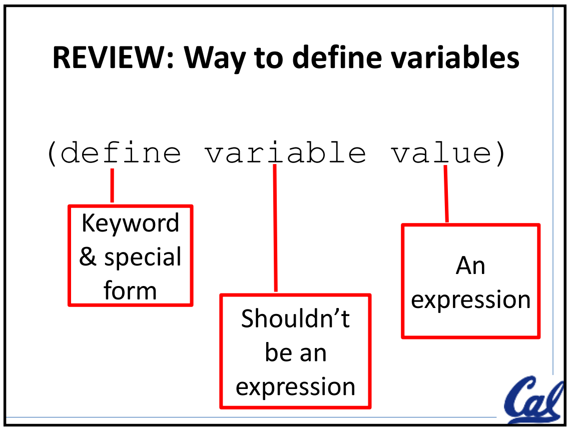
```
STk> (define pi 3.14)
pi
STk> pi
3.14
STk> 'pi
pi
STk> (+ pi 7)
0.14
```

Make a variable

Now no error!

The word pi is still different

Can be used in expressions



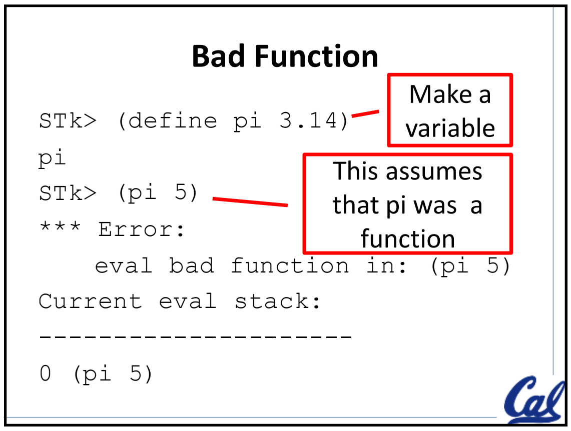
REVIEW: Way to define variables

```
(define variable value)
```

Keyword & special form

Shouldn't be an expression

An expression

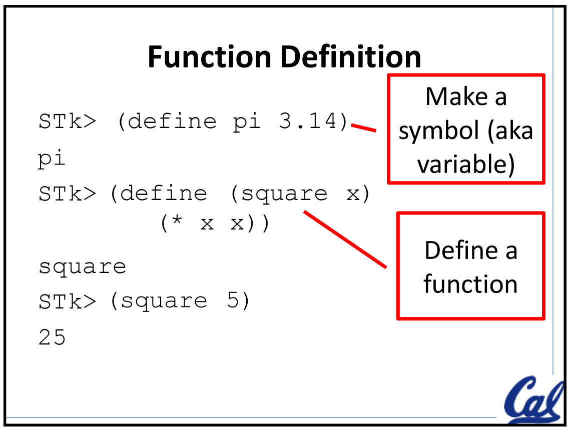


Bad Function

```
STk> (define pi 3.14)
pi
STk> (pi 5)
*** Error:
    eval bad function in: (pi 5)
Current eval stack:
----------------------
0 (pi 5)
```

Make a variable

This assumes that pi was a function

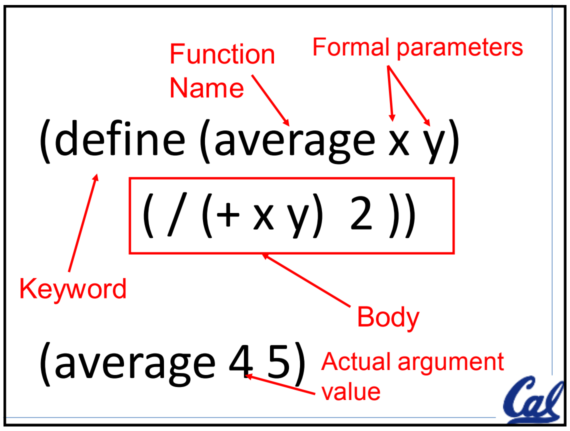


Function Definition

```
STk> (define pi 3.14)
pi
STk> (define (square x)
        (* x x))
square
STk> (square 5)
25
```

Make a symbol (aka variable)

Define a function



```
(define (average x y)
  (/ (+ x y) 2 ))

(average 4 5)
```

Function Name

Formal parameters

Keyword

Body

Actual argument value

Defining plural

```
(define (plural wd)
    (word wd ’s))
```

Implicitly returns last thing

Predicates

- Predicates are procedures that return `#t` or `#f`
  - by convention, their names end with a "?"

odd?	`(odd? 3)` ➔ `#t`
even?	`(even? 3)` ➔ `#f`
vowel?	`(vowel? 'a)` ➔ `#t`
	`(vowel? (first 'fred))` ➔ `#f`
sentence?	`(sentence? 'fred)` ➔ `#f`

Defining Plural (try 2)

```
(define (plural wd)
    (if (equal? (last wd) ’y)
         (word (bl wd) ’ies)
         (word wd ’s)
    )
)
```

IF & COND Statements

```
(if  <predicate>
     <true case>
     <false case>)
(cond
     (predicate1   return_expression1)
     (predicate2   return_expression2)
     (else         return_expression3))
```

Try It

Write a better `plural` function using `cond`

- Works for “fox/foxes” and “wolf/wolves”

```
(cond
     (predicate1       return_expression1)
     (predicate2       return_expression2)
     (else             return_expression3))
```

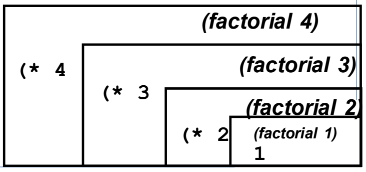
Factorial (Recursion Review)

- 10! = 10*9! (recursive case) x! = x*(x-1)!
- 1! = 1 (base case)
- 0! = 1 (base case)

```
(define (factorial x)
  (if (< x 2)
     1
     (* x (factorial (- x 1)))))
```

What is (factorial 4)?

```
(define (factorial x)
  (if (< x 2)
     1
     (* x (factorial (- x 1)))))
```



All Recursive Procedures Need

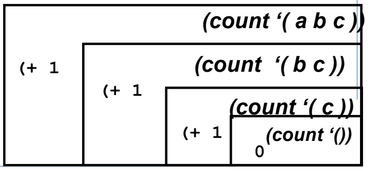
1. Base Case (s)
   - Where the problem is simple enough to be solved directly
2. Recursive Cases (s)
   1. **Divide the Problem (Make the problem Smaller!)**
      - into one or more smaller problems
   2. **Invoke the function**
      - Have it call itself recursively on each smaller part
   3. **Combine the solutions**
      - Combine each subpart into a solution for the whole

Try It!

- Write `count` that takes in a `sentence` and counts the words in the `sentence`.

Count the number of words in a sentence

```
(define (count sent)

  (if (empty? sent)   ;no more?

     0                  ;base case: return 0

     (+ 1
       (count (bf sent))) ;recurse on the
                          ; rest of sent

  ))

>(count '(a b c))
```

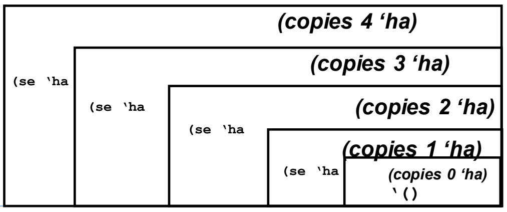


Try It!

- Write `copies` that takes in a `word` and a variable `n` and repeats the word `n` times in a `sentence`.

Copies

```
(define (copies n wd)
  (if (< n 1)
     '()
     (sentence wd (copies (- n 1) wd))))
```


(copies 4 'ha)
(se 'ha
(copies 3 'ha)
(se 'ha
(copies 2 'ha)
(se 'ha
(copies 1 'ha)
(se 'ha
(copies 0 'ha)
'()