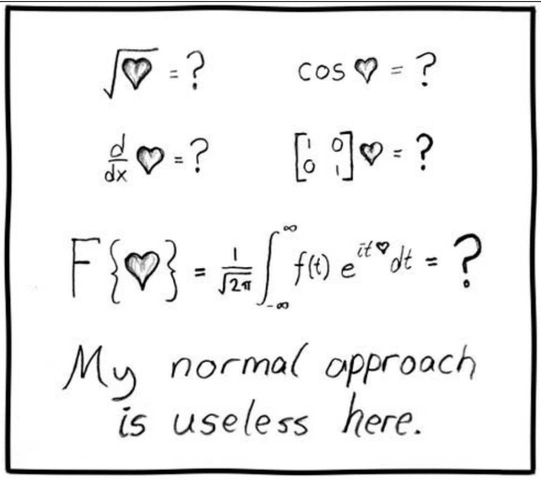

$\sqrt{♡} = ?$
$\cos ♡ = ?$
$\frac{d}{dx} ♡ = ?$
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} ♡ = ?$
$F\{♡\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)\, e^{it♡} dt = ?$
My normal approach is useless here.

The Value of Everything & Procedure Practice

One-Slide Summary

- In Scheme, expressions **evaluate** to values. Five **evaluation rules** describe this process.
- **Lambda** means "make a function". A lambda expression specifies the formal parameter and the function body.
- Evaluating a **function application** involves evaluating the function, finding its body, replacing the formal parameters with the evaluated actual arguments, and evaluating the result.

Lecture Outline

- Survey Responses
- Evaluation Rules
  - Lambda
- Problem Set 1
  - Decent Scheme

Lab and Office Hours

- **Structured Lab Hours**
  - Wed 7-8, Wed 8-9
- **Staffed Lab Hours** (Small Hall)
  - Mon 5-6, Tue 3:15-4:15, Thu 5-6
  - Sun 3-4 (if requested by that Friday)
- Office and **Lab Hours** (Small Hall)
  - Mon 2-3, Tue 3-4, Wed 1-2
  - Tue 11-12 (OLS 219)
- 56/69 can make labs, 64/69 can make office

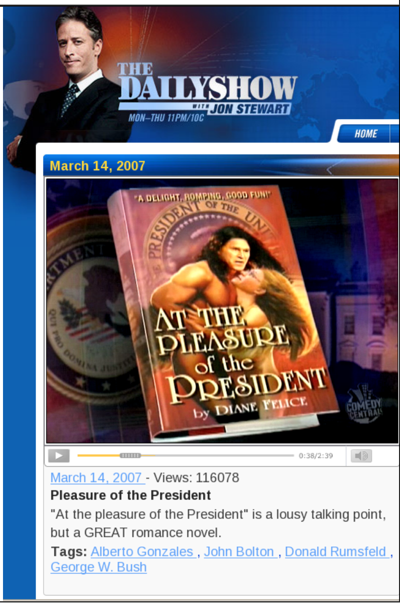
15 Minute No-Show Rule!

Survey Answers

- 59 want "before Spring break"
  - thus Exam 1 due before Spring Break
- 46 want Bodos
- Average Year of CS 150 students: 2.6
- ~33 have previously programmed (mostly Java)
- ~35 unlikely to major in CS, ~29 likely to

The Forum!

- Your questions for me are answered on the forum.
- Any questions right now?


THE DAILYSHOW WITH JON STEWART
MON–THU 11PM/10C
HOME
"A DELIGHT, ROMPING, GOOD FUN!"
AT THE PLEASURE of the PRESIDENT by DIANE FELICE
COMEDY CENTRAL
0:38/2:39

March 14, 2007

March 14, 2007 - Views: 116078

Pleasure of the President

"At the pleasure of the President" is a lousy talking point, but a GREAT romance novel.

Tags: Alberto Gonzales , John Bolton , Donald Rumsfeld , George W. Bush

Who Are You?

- I am an EMT, hard of hearing, an RA, gluten intolerant, a diabetic, tall for an Asian girl, a twin, a fan of anecdotes, a violin-player, a volleyball player, I still have a baby tooth, born on leap year day, a Tolkien fan, afraid of being struck by candy, nervous, a writer of music and screenplays, and is it bad if I get most of your jokes?
  - CS 150 Gestalt Student, Spring 2009

Problem Set 1

- Scheme's **Evaluation Rules** tell you how to find the value of any expression.
- Questions 1 and 2 ask you to evaluate Scheme expressions in your mind
  - This is a popular exam question.
- Without Evaluation Rules: guesswork
- Once you know the Evaluation Rules, you can answer without any guessing!

Evaluation Rules

- **Primitives** (* 55 66)
  - Evaluate to their pre-defined values
- **Names** (+ x 2)
  - Evaluate to the value associated with that name
- **Application** (square-root 144)
  - Eval all sub-expressions. Apply the value of the first (a function) to the values of the others.
- **Lambda** (lambda (x) (* x x x))
  - Evaluates to a function with parameters and body
- **If** (if (< 3 5) 99 11)
  - Eval predicate. If #f, eval second option. Otherwise, eval the first option.

Primitive Examples

55

-88

#t

#f

+

What do these evaluate to?

Primitive Examples

55	-->	55
-88	-->	-88
#t	-->	true (#t)
#f	-->	false (#f)
+	-->	primitive addition

Name Examples

(define x 55)

(define y 66)

x

y

z

What do these evaluate to?

Name Examples

(define x 55)

(define y 66)

x --> 55

y --> 66

z --> *reference to undefined identifier: z*

Application Examples

(sqrt 16)

(abs -5)

(string-length “Hi”)

(+ 1 2)

(+ 1 2 3)

(+ 1)

What do these evaluate to?

Application Examples

(sqrt 16)	--> 4
(abs -5)	--> 5
(string-length “Hi”)	--> 2
(+ 1 2)	--> 3
(+ 1 2 3)	--> 6
(+ 1)	--> 1

Liberal Arts Trivia: Antropology

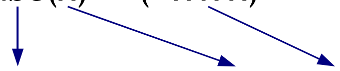
- This American cultural anthropologist is famous for her studies of Samoa and her reports about the purportedly healthy attitude towards sex in South Pacific and Southeast Asian traditional cultures, which influenced the women's liberation movement (e.g., by claiming that females dominated in Chambri and Papau New Guinea without problems). Five years after she died, her work was challenged by Derek Freeman.

Liberal Arts: Slavic Folklore

- This witch-like character in Slavic folklore lives in a walking house with chicken feet (but no windows and no doors), flies around on a giant mortar, and kidnaps (presumably to eat) small children. Modest Mussorgsky's *Pictures at an Exhibition*, a piano suite composed in 1874, features "The Hut on Bird's Legs" as its penultimate movement.

Lambda

- **Lambda** means “make a function”.
- Consider: cube(x) = x * x * x
- Scheme-y: cube(x) = (* x x x)



- Lambda: cube = (lambda (x) (* x x x))
- Pure Scheme:

(define cube (lambda (x) (* x x x)))

Anatomy Of A Function

- **(define cube (lambda (x) (* x x x)))**

formal parameters **function body**

- **(cube 5)**

function application **actual arguments**

- To **evaluate a function application**, replace it with the function body, and then replace every formal parameter with its corresponding actual argument.
- (cube 5) -> (* x x x) -> (* 5 5 5) -> 125

Lambda Examples

(define cube (lambda (x) (* x x x)))
(define foo (lambda (p q) (+ p q)))
(define bar (lambda (a b c) (* a c)))

(cube 3)
(foo 5 6)
(bar 4 5 6)
(foo (cube 3) 1)

What do these evaluate to?

Lambda Examples

(define cube (lambda (x) (* x x x)))
(define foo (lambda (p q) (+ p q)))
(define bar (lambda (a b c) (* a c)))

(cube 3) --> (* 3 3 3)->27
(foo 5 6) --> (+ 5 6) -> 11
(bar 4 5 6) --> (* 4 6) -> 24
(foo (cube 3) 1) --> ... -> 28

Lambda Lambda Lambda

- Consider these two functions:
  - **(define cube (lambda (x) (* x x x)))**
  - **(define cube (lambda (y) (* y y y)))**
- Are they different?
- Consider:
  - **(define nail (lambda (x y) (+ x y)))**
  - **(define polish (lambda (y x) (/ y x)))**
- What is:
  - **(polish (nail 6 4) 2)**

Do this now on paper!

Sally Hansen does Lambda

(define nail (lambda (x y) (+ x y)))
(define polish (lambda (y x) (/ y x)))
(polish (nail 6 4) 2)

- This is a call to **polish** with tricky arguments.
- Recall the rule: evaluate the arguments first.
  - Argument 1: (nail 6 4) -> (+ x y) -> (+ 6 4) -> 10
  - Argument 2: 2 -> 2
- Now take **polish**'s body, and replace the formal parameters with the actual arguments:
  - (/ y x) -> (/ 10 2) -> 5

Why not (/ 2 10) ?

If Examples

(if #t “yes” “no”)
(if #f “yes” “no”)
(if (< 3 5) “ant” “bat”)
(if (< 5 3) “cat” “dog”)
(if “x” “y” “z”)
(if (if 11 #f #t) 22 33)

What do these evaluate to?

If Examples

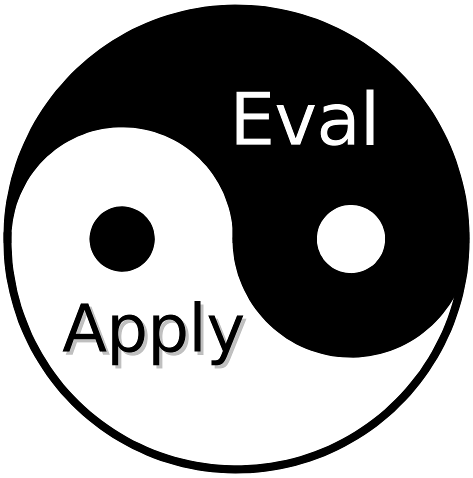
```
(if #t “yes” “no”)          -> “yes”
(if #f “yes” “no”)          -> “no”
(if (< 3 5) “ant” “bat”)  -> “ant”
(if (< 5 3) “cat” “dog”)  -> “dog”
(if “x” “y” “z”)            -> “y”
(if (if 11 #f #t) 22 33)    -> 33
```

Scheme Trickery

- (100 + 100)
  - Error: The expression in the first position must be a function (or something special like **if**). 100 is not a function.
- **(if (not “batterie”) “fouetté” “plié”))**
  - “plié”. **(not “batterie”)** returns **#f**, because **“batterie”** is not **#f**.
    - (define (not v) (if v #f #t))
- Does (if *X* #t #f) always equal *X* ?
  - Yes for #t, #f, (< 3 5), (> 5 6).
  - No for 3, 17, “hello”.

Evaluating expressions and **Applying functions** are defined in terms of each other.

Without Eval, there would be no Apply, Without Apply there would be no Eval!



Now You Know All of Scheme!

- Once you understand **Eval** and **Apply**, you can understand all Scheme programs!
- Except:
  - There are many primitives, and you need to know their predefined meaning.
  - There are a few more special forms (like **if**).
  - We have not define the evaluation rules precisely enough to unambiguously understand all programs (e.g., what does “value associated with a name” mean?).

Now On To Problem Set 1

- Smooth transition ...

brighter?

```
(define brighter? (lambda (color1
color2) (if (> (+ (get-red color1)
(get-green color1) (get-blue color1)
) (+ (get-red color2)  (get-green
color2) (get-blue color2)) #t
#f)))
```

Is this correct?

Maybe...but very hard to tell. Your code should appear in a way that reveals its structure

```
(define brighter?
  (lambda (color1 color2)
    (if (> (+ (get-red color1)
              (get-green color1)
              (get-blue color1))
           (+ (get-red color2)
              (get-green color2)
              (get-blue color2))
     #t #f)))
```

Use [Tab] in DrScheme to line up your code structurally!

Brighter **brighter?**?

```
(define brighter?
   (lambda (color1 color2)
      (> (+ (get-red color1)
            (get-green color1)
            (get-blue color1))
         (+ (get-red color2)
            (get-green color2)
            (get-blue color2)))))
```

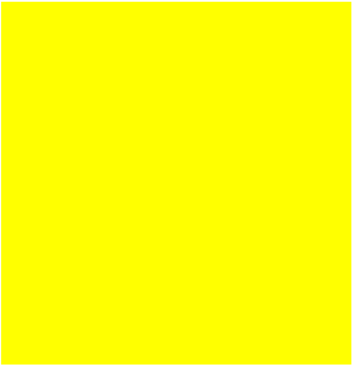
What can we do about this duplicated code?

Brighter **brighter?**?

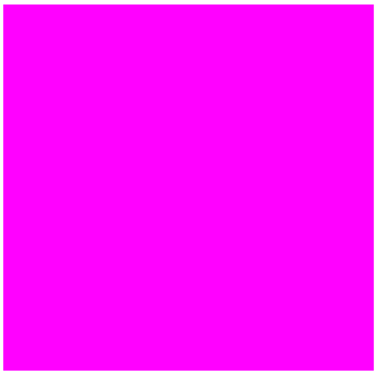
```
(define brightness
   (lambda (color)
       (+ (get-red color)
          (get-green color)
          (get-blue color))))

(define brighter?
   (lambda (color1 color2)
      (> (brightness color1)
         (brightness color2))))
```

Believable **brighter?**?

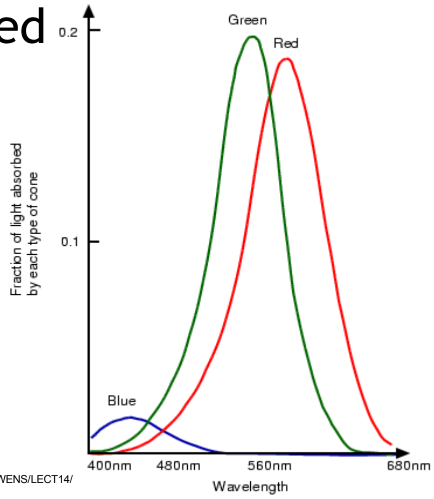


(make-color 255 255 0)



(make-color 255 1 255)

Color Absorbed



http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT14/

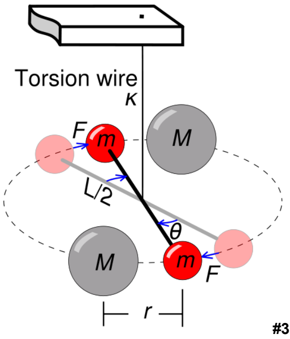
Cognitive Scientist’s Answer

```
(define brightness
   (lambda (color)
       (+ (* 0.299 (get-red color))
          (* 0.587 (get-green color))
          (* 0.114 (get-blue color)))))

(define brighter?
   (lambda (color1 color2)
      (> (brightness color1)
         (brightness color2))))
```

Liberal Arts Trivia: Physics

- This 1797 torsion balance experiment, sometimes called “weighing the earth”, was the first to measure the force of gravity between masses in the laboratory, and the first to yield accurate values of the gravitational constant and thus the mass of the Earth.



Liberal Arts Trivia: Drama

- This classical Athenian tragedy by Sophocles, first performed in BC 429, is widely considered a supreme masterpiece of the art of Drama. The Oracle at Delphi tells the protagonist that he is doomed to marry his mother and kill his father. He goes on to do so, but not before solving the riddle of the sphinx: What is the creature that walks on four legs in the morning, two legs at noon, and three in the evening? Name the play *and* answer the riddle.

What should you do if you can’t get your code to work?

- Keep trying: think of alternate approaches
- Get help from the TAs and your classmates
- But, if it's too late for that ...
  - **In your submission, explain what doesn’t work and as much as you can what you think is right and wrong**
- If you get less than 50% on the automatic adjudication part, the TAs will look over your source and give partial credit.

Evaluation Rules

- A formal review and study guide follows ...

Primitive Expressions

Expression ::= *PrimitiveExpression*

PrimitiveExpression ::= ***Number***

PrimitiveExpression ::= **#t** | **#f**

PrimitiveExpression ::= ***Primitive Procedure***

Evaluation Rule 1: Primitive. If the expression is a *primitive*, it evaluates to its pre-defined value.

```
> +
#<primitive:+>
```

Name Expressions

Expression ::= *NameExpression*

NameExpression ::= ***Name***

Evaluation Rule 2: Name. If the expression is a *name*, it evaluates to the value associated with that name.

```
> (define two 2)
> two
2
```

Definitions

Definition ::= **(define** ***Name*** *Expression***)**

Definition Rule. A definition evaluates the *Expression*, and associates the value of *Expression* with *Name*.

```
> (define dumb (+ + +))
+: expects type <number> as 1st argument, given: #<primitive:+>;
other arguments were: #<primitive:+>
> dumb
reference to undefined identifier: dumb
```

Application Expressions

Expression ::= *ApplicationExpression*

ApplicationExpression ::= (*Expression MoreExpressions*)

MoreExpressions ::= ε | *Expression MoreExpressions*

Evaluation Rule 3: Application. To evaluate an application expression:

a. Evaluate all the subexpressions;

b. Then, **apply** the value of the first subexpression to the values of the remaining subexpressions.

Rules for Application

- **Primitive.** If the procedure to apply is a *primitive*, just do it.
- **Constructed Procedure.** If the procedure is a *constructed (lambda) procedure*, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.

Constructing Procedures: Lambda

Expression ::= *ProcedureExpression*

ProcedureExpression

::= **(lambda (***Parameters***)** *Expression***)**

Parameters ::= ε | ***Name*** *Parameters*

Evaluation Rule 4: Lambda. Lambda expressions evaluate to a procedure that takes the given *Parameters* as inputs and has the *Expression* as its body.

Applying Constructed Procedures

Application Rule 2: Constructed Procedure. If the procedure is a *constructed (lambda) procedure*, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.

Applying Constructed Procedures

Application Rule 2: Constructed Procedure. If the procedure is a *constructed procedure*, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.

```
> ((lambda (n) (+ n 1)) 2)
```

↓ Evaluation Rule 3a: evaluate the subexpressions

((lambda (n) (+ n 1)) 2)

↓ Evaluation Rule 3b, Application Rule 2

(+ 2 1)

↓ Evaluation Rule 3a, 3b, Application Rule 1

3

Evaluation Rule 5: If

$$\mathit{Expression} ::= (\textbf{if}\ \mathit{Expression}_{\text{Predicate}}$$
$$\mathit{Expression}_{\text{Consequent}}$$
$$\mathit{Expression}_{\text{Alternate}})$$

Evaluation Rule 5: If. To evaluate an if expression:

- Evaluate the predicate expressions.
- If it evaluates to #f, the value of the if expression is the value of alternate expression. Otherwise, the value of the if expression is the value of consequent expression.

Lambda Example: Tautology Function

```
(lambda       make a procedure
 ()           with no parameters
 #t)          with body #t
```

```
> ((lambda () #t) 150)
#<procedure>: expects no arguments, given 1: 150
> ((lambda () #t))
#t
> ((lambda (x) x) 150)
150
```

Homework

- (In theory) You now know everything you need for PS1, PS2, PS3 and PS4 ...
- Problem Set 1 due Monday January 26