Prelim 1 Review Session

Start now!

Topics

Recursion (e.g. evaluation tree as in Fibonacci)

Recursion (iteration)

Short circuit evaluation → and / or

Lexical scoping rules (free variables)

Higher Order Functions

Define

Define (special form)

Defining bindings (“variables”):

```
(define var_name_here expression_goes_here)
```

Example:

```
(define e 2.718)
```

Define (special form)

Defining procedures (functions):

```
(define (func_name_here <parameter names separated by spaces>)
    expression_goes_here)
```

Example:

```
(define (square x)
        (* x x))
```

Let Statements

Let Format

```
(define (someFunction param1 …)
    (let ((x1 <expr1>) (x2 <expr2>))
        Rest_of_function_body_goes_here))
```

() to close let

() to close statements

expr1 , expr2 , etc. evaluate to something which get bound to x1, x2, etc. respectively

You can have as many bindings as you’d like

However, with regular let in *this* example you would not be able to use x1 in expr2

Let Examples

```
(define (area-of-circle r)
    (let ((pi 3.14))
        (* pi (* r r))))
```

In the first example, we use a let to define a constant: pi = 3.14

```
(define (distance x1 x2 y1 y2)
  (let ((squarex (* (- x2 x1) (- x2 x1)))
         (squarey (* (- y2 y1) (- y2 y1))))
         (sqrt(+ squarex squarey))))
```

In this one, the let is used to define expressions using the values given in the function

If Statements

If Construction

```
(if <predicate>
    then_expression_goes_here
    else_expression_goes_here)
```

Example:

```
(define (absolute x)
    (if (< x 0)
        (- x)
        x))
```

Incorrect If Syntax

```
(define (absolute x)
    (if (< x 0)
        (- x)
        (else x)))
```

An if statement does not use an else! This is only used in a cond.

```
(define (absolute x)
        (if (< x 0))
        (- x)
        x)
```

This if statement is *closed* before it provides an output to the condition. We don't like that. We want outputs.

When NOT to Use If

```
(define (equal5? x)
    (if (= x 5)
        #t
        #f))
```

Instead, do this:

```
(define (equal5? x)
    (= x 5))
```

```
(define (equal5-10? x)
    (if (= x 5)
        #t
        (if (= x 10)
            #t
            #f)))
```

```
(define (equal5-10? x)
    (or (= x 5) (= x 10))))
```

Cond Statements

Cond Formatting

```
(cond ((guard1) expression1)
      ((guard2) expression2)
      ...
      (else expression3))
```

When you see a question that looks like this, you typically need cond:

$$GCD(m,n) = \begin{cases} n & \text{if } m = n, \\ GCD(m-n,n) & \text{if } m > n, \\ GCD(m,n-m) & \text{if } n > m. \end{cases}$$

Boolean Expressions

And

Or

Logical Composition Operators and some other stuff

- (and a1 a2 … an) → takes in any number of booleans evaluated left to right, stops if it finds #f***
- (or a1 a2 … an) → takes in any number of booleans, evaluated left to right, stops if it finds #t ***
- (not a1) → takes in ONE boolean (#t #f or an expression that evaluates to #t or #f) ***

***Something I won't get into here, but if you're interested see Racket documentation for how it will evaluate with numeric values and other expressions or try it on your own! (e.g. (and #t 5) vs. (and 5 #t), (or #t 5) vs. (or 5 #t), (not 5), etc.)

One thing to note...

- (= a1 … an) → numeric expression, does *not* take booleans
  - (e.g. (= #t #t) will give you an error:"contract violation expected: number? given: #t…" Think this is on par with (> #t #t) just doesn't make sense)

=, >, <, >=, <= are all numeric comparisons

And Truth Table

```
(and Value1 Value2)
```

Value1	Value2	Output
#t	#t	#t
#t	#f	#f
#f	#t	#f
#f	#f	#f

Or Truth Table

```
(or Value1 Value2)
```

Value1	Value2	Output
#t	#t	#t
#t	#f	#t
#f	#t	#t
#f	#f	#f

Not Truth Table

```
(not Value1)
```

Value1	Output
#t	#f
#f	#t

Lexical Scope

Expected Output #1

```
(define x 100)
(define (whatstheoutput? y) (+ x 100))
(whatstheoutput? 3)
; answer on next animation click
> 200
Why?
(define x 100)------------------------->        x→100
(define (whatstheoutput? y) (+ x 100))
(whatstheoutput? 3) -------------------->       y→ 3
Call to (whatstheoutput? 3) looks for the binding for x in the
environment → (+ x 100) → (+ 100 100)
```

Expected Output #2

```
(define x 100)
(define (whatstheoutput2? x) (+ x 100))
(whatstheoutput2? 3)
; answer on next animation click
> 103
```

Why?

```
(define x 100)
(define (whatstheoutput2? x) (+ x 100))
(whatstheoutput? 3)
```

Call to (whatstheoutput2? 3) looks for the binding for x in the environment → (+ x 100) → (+ 3 100)

x→100

x→ 3

Expected Output #3

```
(define x 100)

(define (whatstheoutput3? x)
  (let ((x 2))
        (+ x 100)))
(whatstheoutput3? 3)
```

x ⟶ 100

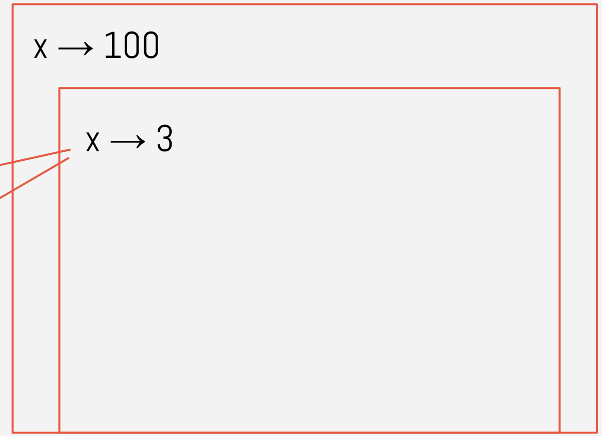
Expected Output #3

```
(define x 100)

(define (whatstheoutput3? x)
  (let ((x 2))
        (+ x 100)))
(whatstheoutput3? 3)
```

call to whatstheoutput3?

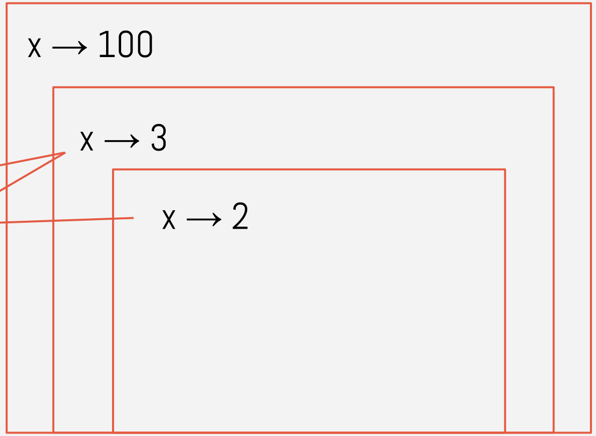
Expected Output #3

```
(define x 100)


(define (whatstheoutput3? x)
  (let ((x 2))
        (+ x 100)))
(whatstheoutput3? 3)
```



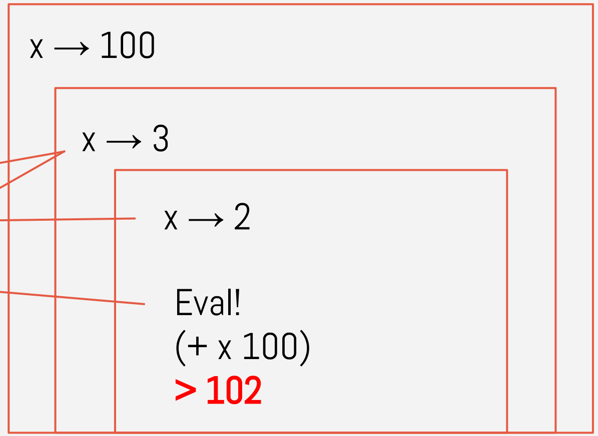
Expected Output #3

```
(define x 100)


(define (whatstheoutput3? x)
  (let ((x 2))
        (+ x 100)))
(whatstheoutput3? 3)
```



Expected Output #3

```
(define x 100)


(define (whatstheoutput3? x)
  (let ((x 2))
        (+ x 100)))
(whatstheoutput3? 3)
```



Expected Output #3

```
(define x 100)


(define (whatstheoutput3? x)
  (let ((x 2))
        (+ x 100)))
(whatstheoutput3? 3)
x
```

x ⟶ 100

Eval!

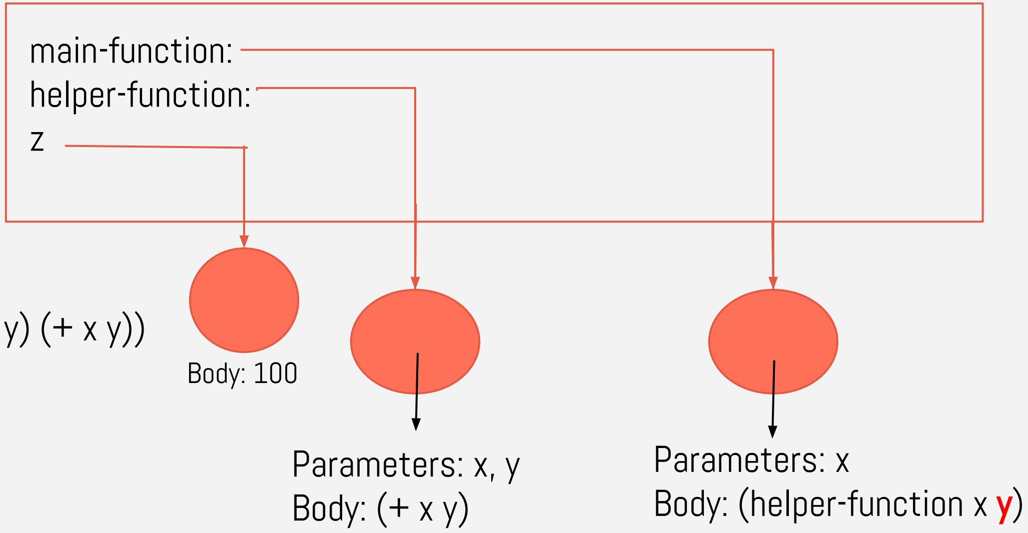
x

> 100

- The evaluation of x OUTSIDE of whatstheoutput3?'s function body here cannot see the local bindings INSIDE of whatstheoutput3?'s function body!!!
- Think what is a global and what is a local binding!

Commonly Seen Error.......

Global environment →



CODE:

```
(define z 100)
(define (helper-function x y) (+ x y))

(define (main-function x)
  (helper-function x y))

(main-function 3)
```

If I call (main-function 3) IT WILL GIVE AN ERROR:
"y: undefined; cannot reference an identifier before its definition"
because helper-function has a definition of y in **ITS** environment if I were to call e.g. (helper-function 1 2) that's valid, **but MAIN-FUNCTION DOES NOT HAVE A DEFINITION FOR Y UPON THE INVOCATION (calling) OF (main-function 3)!!!!!**

Example Problems

Even? Function

Create a function called *even* that for a given number n, (even n) returns true if n is even and false if it’s not.

solution

What is the even function doing?

- Checking to see if an input n is divisible by 2 or not

How can we implement this?

- Use modulo to check if n is fully divisible by 2, if it is, then return true

```scheme
(define (even n)
  (cond ((zero? (modulo n 2)) #t)
        (else #f)))
```

*the built-in zero? function is replaceable by a simple
((= 0 (modulo n 2)) #t) as well

Defining Pi

Just DO IT!!

Define a Scheme variable for the constant pi with the numeric value of the given expression:

$$\pi = \sqrt[4]{\frac{2143}{22}} = \left(\frac{2143}{22}\right)^{\frac{1}{4}}$$

You may use the built-in Scheme function *expt* which computes b^e.

solution

What is the pi function doing?

- Defining a Scheme variable called *pi* with the given equation

How can we implement this?

- Define *pi* as the solution to the equation using the built-in *sqrt* function to help

```
(define pi (expt (/ 2143 22) (/ 1 4)))
```

Function with a Given Formula

Define a function f, which given a positive number n, returns:

$$\frac{\left(1+\sqrt{5}\right)^n-\left(1-\sqrt{5}\right)^n}{2^n\sqrt{5}}$$

solution

What is the f function doing?

- Giving us the output of the given function depending on the input n

How can we implement this?

- We state n as the parameter for the function
- We can use Scheme's built-in functions *expt* and *sqrt*

```
(define (f n)
  (/  (- (expt (+ 1 (sqrt 5)) n)
          (expt (- 1 (sqrt 5)) n))
      (* (expt 2 n) (sqrt 5))))
```

Double Factorial

The double factorial function is a variant on the classic factorial and is defined as follows:

$$n!! = \begin{cases} \prod_{i=1}^{\frac{n+1}{2}} (2 \cdot i - 1) & \text{if } odd(n) \\ \prod_{i=1}^{\frac{n}{2}} (2 \cdot i) & \text{if } even(n) \end{cases}$$

Create a function called (dfact n) that returns the double factorial of n.

Things to note

- Double factorial is not the same as doing (3!)! = 6!
    - 3!! = 3*1 = 3
    - It is the process of multiplying every *other* number that comes after n
        - Ex 15!! = 15*13*11*9*7*5*....

$$n!! = \begin{cases} \prod_{i=1}^{\frac{n+1}{2}} (2 \cdot i - 1) & \text{if } odd(n) \\ \prod_{i=1}^{\frac{n}{2}} (2 \cdot i) & \text{if } even(n) \end{cases}$$

$$n!! = \begin{cases} \prod_{i=1}^{\frac{n+1}{2}}(2 \cdot i - 1) & \text{if } odd(n) \\ \prod_{i=1}^{\frac{n}{2}}(2 \cdot i) & \text{if } even(n) \end{cases}$$

How do we approach this?

- We need 2 helper functions, one for if n is odd, one for if n is even
- How do we implement this?
- Two different formulas for even and odd
- (Recursive product)
    - What should the base case be for each helper function?

Solution Code:

```
(define (dfact n)
  (define (helper-even x i)
    (if (> i x)
        1
        (* 2 i (helper-even x (+ i 1)))))

  (define (helper-odd x i)
    (if (> i x)
        1
        (* (- (* 2 i) 1) (helper-odd x (+ i 1)))))

  (cond ((= 0 n) 1)
        ((zero? (modulo n 2)) (helper-even (/ n 2) 1))
        (else (helper-odd (/ (+ n 1) 2) 1))))
```

Another Way to Approximate Pi

In 1666, Newton used a geometric construction to derive a formula for pi. Using Euler's convergence improvement transformation gives the following:

$$\frac{\pi}{2} = \sum_{i=0}^{n} \left(\frac{i!}{(2i + 1)!!} \right)$$

Define a scheme function, named (pi-approx n), for approximating pi using n terms in this formula. Assume that you are given the factorial function as: (fact n) and double factorial function (dfact n) from the previous problem.

What is the function doing?

- Evaluating to a number (approximation of π)
- How?
  - Sum of a series of terms
  - So, we can use recursion
- How does a series sum translate to recursion?
  - Each recursive call computes a term...
  - ...and adds it to the rest of the terms.
- What changes from one call to the next? How do we ensure it will approach a base case?
  - The parameter n will decrement towards n = 0 and compute terms in reverse order.
- What's the base case?
  - i = 0 is the first defined term in the sum. (and the last one we'll compute) so the base case occurs when n = 0.

```scheme
(define (pi-approx n)
    (if (= n 0)
        1
        (+ (/ (fact n) (dfact (+ (* 2 n) 1)))
           (pi-approx (- n 1)))))
```

```
Let's test it!
> (pi-approx 50)
```

$$1\frac{20934424700375306622725071181596499775163}{36675822386463334341759972408679909683525}$$

```
> (exact->inexact (pi-approx 50))
1.5707963266505798
What?
```

$$\frac{\pi}{2} = \sum_{i=0}^{n}\left(\frac{i!}{(2i+1)!!}\right)$$

```
(define (pi-approx n)
    (define (pi-approx-helper i)
        (if (= i 0)
            1
            (+ (/ (fact i) (dfact (+ (* 2 i) 1)))
               (pi-approx-helper (- i 1)))))
    (* 2 (exact->inexact (pi-approx-helper n))))
```

```
> (pi-approx 50)
3.141592653589793
> (pi-approx 30)
3.141592653011596
> (pi-approx 20)
3.1415922987403397
> (pi-approx 7)
3.137129537129537
```

Piecewise Function

Piecewise Function

Consider the following function:

$$f(x) = \begin{cases} x^2 - 4 & \text{if } x < -2, \\ -x^2 + 4 & \text{if } -2 \leq x \leq 2, \\ x^2 - 4 & \text{if } x > 2. \end{cases}$$

Define a function piecewise so that (piecewise x) returns f(x).

The first thing we notice is that there are multiple conditions in this problem.

Identify the following conditions:

- $x < -2$
- $-2 \le x \le 2$
- $x > 2$

What can we use for this…

A COND STATEMENT

```
(define (piecewise x)
  (cond
    ((< x -2) (- (expt x 2) 4))
    ((and (>= x -2) (<= x 2)) (+ (* (expt x 2) -1) 4))
    ((> x 2) (- (expt x 2) 4))))
```

Arc Tangent Taylor Series

Gregory's series is an infinite Taylor series expansion for the inverse tangent function. The series is:

$$\int_0^x \frac{du}{1+u^2} = \arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

Write a Scheme function (gregory x k) that computes an approximation for arctan x containing the first the first k + 1 terms in the series above, i.e., it computes:

$$\arctan(x) \approx \sum_{i=0}^{k} (-1)^i \frac{x^{(2i+1)}}{2i+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots \pm \frac{x^{2k+1}}{2k+1}$$

Feel free to use the expt function which computes b^e. Hint: using a let statement can save you some ink.

Solution (A Quick Note)

$$\arctan(x) \approx \sum_{i=0}^{k} (-1)^i \frac{x^{(2i+1)}}{2i+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots \pm \frac{x^{2k+1}}{2k+1}$$

This equation uses the variable i, however in the world of recursive programming, we don’t actually need this variable. This is because our function takes in the value for k, and this equation must be done for each value between 0 and k. Thus to perform this for every value, we will recursively subtract k by 1, and calculate what we need each time.

Solution

What is the first thing we look for?

$$\arctan(x) \approx \sum_{\boxed{i=0}}^{k} (-1)^i \frac{x^{(2i+1)}}{2i+1} = \boxed{x} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots \pm \frac{x^{2k+1}}{2k+1}$$

The base case!!! -> When the given k is 0, return x.

Again, not i. This is because we will recursively decrease k and calculate this fraction *for each value of k* (the original meaning of i) until it gets to 0. The variable i is just mathematical notation

Solution Side Note. When do I need a helper??

Helper functions are all about the parameters. If you find that you need more parameters to make this all work, that's when you need a helper.

Solution Side Note. When do I need a helper??

Consider the Catalan Numbers:

$$C_n = \prod_{k=2}^{n} \frac{n+k}{k}$$

The n-th Catalan number depends on BOTH the current value of k and the limit that k goes to, n.

Thus, you need to remember both n and k in order to calculate any term. This requires two parameters, n and k, for the function. However, (catalan n) only takes in one. Thus you need to either write a function outside of (catalan n) called (catalan-helper n k) OR, write (catalan-helper k) INSIDE of (catalan n).

Solution. Writing the base case.

Begin by writing the definition and the base case:

```
(define (gregory x k)
  (cond ((= k 0) x)
```

If is also valid. I just like cond statements.

Solution. Calculating the fraction

$$(-1)^i \frac{x^{(2i+1)}}{2i+1}$$

```
(* (expt -1 k)
   (/ (expt x (+ (* 2 k)
                  1))
      (+ (* 2 k)
         1)))
```

Solution. Calculating the fraction

```
(define (gregory x k)
  (cond ((= k 0) x)
        (else (* (expt -1
                       k)
                 (/ (expt x (+ (* 2 k)
                               1))
                    (+ (* 2 k)
                       1))))))
```

Solution. The recursive call.

$$\arctan(x) \approx \boxed{\sum_{i=0}^{k}} \boxed{(-1)^i \frac{x^{(2i+1)}}{2i+1}} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots \pm \frac{x^{2k+1}}{2k+1}$$

For every fraction we calculate for the current value of k, we need to add that with all of the other fractions.

Solution. The recursive call.

$$\arctan(x) \approx \boxed{\sum_{i=0}^{k}} \boxed{(-1)^i \frac{x^{(2i+1)}}{2i+1}}$$

```
(define (gregory x k)
  (cond ((= k 0) x)
        (else (+ (* (expt -1
                          k)
                    (/ (expt x (+ (* 2 k)
                                  1))
                       (+ (* 2 k)
                          1)))
                 (gregory x (- k 1))))))
```

Splice (Higher Order Function)

1. Let f and g be two functions taking numbers to numbers. Define the function $\text{splice}_{f,g}$ so that

$$\text{splice}_{f,g}(x) = \begin{cases} f(x) & \text{if } x < 0, \\ (1-(3x^2-2x^3))f(x) + (3x^2-2x^3)g(x) & \text{if } 0 \le x \le 1, \\ g(x) & \text{if } x > 1. \end{cases}$$

The splice function smoothly transitions from the function f (on values $x < 0$) to the function g (on values $x > 1$). Write a SCHEME function `splice` so that, given two functions f and g, `(splice f g)` returns the function $\text{splice}_{f,g}$. Note that the value returned by `(splice f g)` should be a *function*.

For example, if $f(x) = \sin(x)$ and $g(x) = 1/(1+x^2)$, the function $\text{splice}_{f,g}$ is graphed in Figure 1 below. The function $3x^2 - 2x^3$, featured in splice, is known as the "smoothstep" function and is frequently used in graphics processing to smoothly interpolate between two functions.

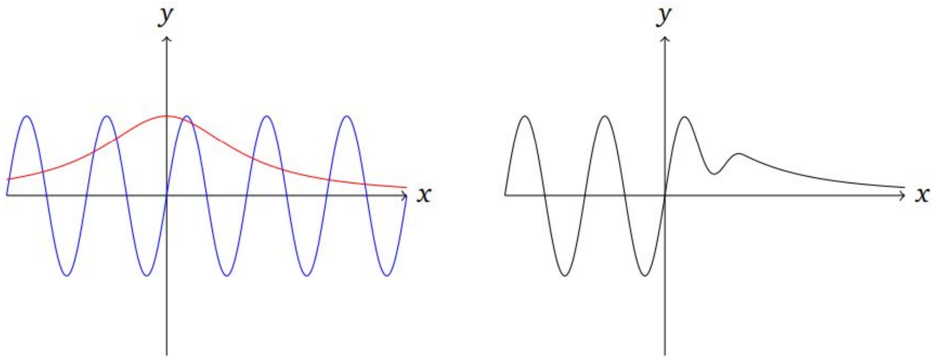


Figure 1: The functions $f(x) = \sin(x)$ and $g(x) = 1/(1+x^2)$ (graphed in blue and red) on the left, and the result when they are spliced together on the right.

A Possible solution

```
(define (splice f g)
  (define (helper x)
    (cond ((< x 0) (f x))
          ((> x 1) (g x))
          (else (+ (* (  - 1
                          (- (* 3 (* x x))
                             (* 2 (* x x x))))
                      (f x))
                   (* (- (* 3 (* x x))
                         (* 2 (* x x x)))
                      (g x))))))
  helper)
```

Questions?