

# CSCI: 4500/6500 Programming Languages

## Functional Programming Languages Part 2: Tutorial



Maria Hybinette, UGA

1

# Tutorial: Scheme

- **Download:**
  - » <http://download.plt-scheme.org/drscheme/>
- **Select Platform and Download links**
- **IDE: Select Appropriate Language:**
  - » “Advance Student” or is a good start
  - » “Essentials of Programming Languages 2nd )
  - » Create a file using an external editor
- **Top: Definitions of Functions**
- **Bottom: Scheme listener, which read input, evaluates and prints out results**

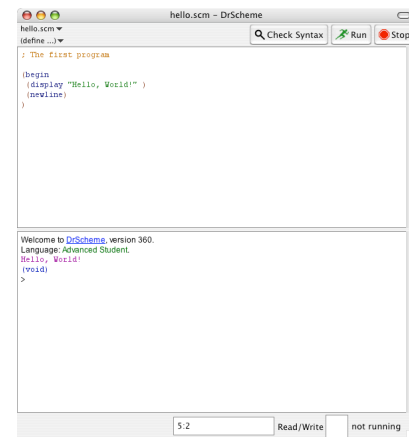
Maria Hybinette, UGA

2

## Simple Introduction

Maria Hybinette, UGA

3



## List Examples

```
> null
empty
> (cons 1 null)
(1)
(list 1)
> (list? null)
#f
true
> (list? (cons 1 2))
#f
> (list? (cons 1 null))
#t
> (list? (cons 1 (cons 2 null)))
#t
> (car (cons 1 (cons 2 null)))
1
> (cdr (cons 1 (cons 2 null)))
(2)
```

Maria Hybinette, UGA

5

## Predicates

- **null?** tests whether it is the empty list e.g.
  - » (null? '(A B)) => #f
  - » (null? '()) => #t
- **list?** takes a single argument and returns #t if its single argument is a list. e.g.,
  - » (list? '(x y)) => #t
  - » (list? 'x) => #f
  - » (list? (1 . 2)) => #f
- **pair?** takes a single argument and returns #t if its single argument is a pair. e.g.,
  - » (pair? '(1 . 2)) => #t
  - » (pair? '(1 2)) => #f
  - » (pair? '()) => #f

Maria Hybinette, UGA

6

## map, apply : Built-in Higher-Order Functions

- **map** applies a function to *sequence of lists*.
  - » `(map + '(2 3 4) '(5 4 3)) => (7 7 7)`
  - » must be as many lists as number of arguments of the function
  - » all lists must be of the same length
- **apply** applies its first argument to its second argument -- a list.
  - » `(apply max '(3 7 2 9)) => 9`

Maria Hybinette, UGA

7



## Searching for members

- **memq, memv and member**
- List procedures:
  - » Check help pages
- `(memq 'a '(a b c))`
- `(memq 'b '(a b c))`
- `(memq 'a (b c d))`



Maria Hybinette, UGA

8

## Consequents: Conditionals Expressions

- `(if condition then-consequent else-alternative)`
- Example:
  - » `(if (< 2 3) 4 5) => 4`
- `(cond (condition1 then-consequent1) (condition2 then-consequent2) . . . (else alternative))`
- Example:
  - » `(cond ((< 3 2) 1) ((< 4 3) 2) (else 3)) => 3`

Maria Hybinette, UGA

9

## List and functions: f(g(x))

- Take the car of the rest
- `(define ccompose (lambda (fg) (lambda (x) (f (g x)))))`
- `((ccompose car cdr) '(1 2 3))`  
->2
- High order function! Takes a function as an argument (or return one)

Maria Hybinette, UGA

10

## Math: Factorials!

- $10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$
- $f(0) = 0! = 1 ; n = 0$
- $f(n) = n * f(n-1) ; n > 0$
- $f(3) = 3 * f(3-1) = 3 * f(2) = 3 * 2 * f(2-1) = 3 * 2 * f(1)$
- Base case in scheme: `(if (= n 0) 1 ?)`
- The rest:
 

```
(define (factorial n)
  (lambda (n)
    (
      if (= n 0)
        1
        (* n (factorial (- n 1)))))
  ))
> (factorial 4)
24
```

Maria Hybinette, UGA

11

## Math: Factorials!

- The rest:
 

```
(define (factorial n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
> (factorial 4)
24
```
- Equivalent (back to lambda):
 

```
(define factorial
  (lambda (n)
    (if (= n 0)
        1
        (* n (factorial (- n 1)))))
  ))
> (factorial 4)
24
```

Maria Hybinette, UGA

12

## Practice: Fibonacci Numbers

- 1 2 3 4 5 6 7
- 1, 1, 2, 3, 5, 8, 13, ...
- Each term after the second is the sum of the preceding two
- Define the recursion:
  - » fib(n) = 1 ; if n = 0 or n = 1
  - » fib(n) = fib(n-1) + fib(n-2) ; otherwise "do this first"
- (+ (fib (- n 1)) (fib (- n 2)))
- (define fib
  - » (lambda (n)
    - (if (= n 0)
      - 0
      - (if (= n 1)
        - » 1
        - » (+ (fib (- n 1)) (fib (- n 2))))

Maria Hybinette, UGA

13

## Bindings: Local Variables

- let, let\* letrec creates local variables
- (let ((var1 exp1)  
(var2 exp2) . . .  
(varn expn))  
body )
- Scope of variable within body (only) of let:
  - » Example:  
>(let ( ( x 2)  
( y 10))  
(+ x y))  
12
- Let\*: allows you do use previously bounded variables when defining a new variable (so the scope is not in the body only)
- letrec: allows recursion

Maria Hybinette, UGA

14

## Sequencing

```
(define display3
  (lambda (arg1 arg2 arg3)
    (begin
      (display arg1)
      (display " ")
      (display arg2)
      (display " ")
      (display arg3)
      (newline))))

(display3 "this" "is" "great!")
```

Maria Hybinette, UGA

15

## Run Scheme from OS prompt

- OS prompt driven:
  - » mzscheme [-r] [hello.scm]
- (load "hello.scm") ; Language Choice
- File of loads

Maria Hybinette, UGA

16

## Characteristics of "Pure" Functional Languages

- No side effects (e.g. no access to global variables)
- No assignment statements
- Often no variables
- Small concise framework
- Simple uniform syntax
- Recursive (that is how we get things done)
- Interpreted

Maria Hybinette, UGA

17

## Learning Scheme

- Practice

Maria Hybinette, UGA

18

## Summary

---

- **Downloading Scheme**
- **Installation & Running**
- **Examples**
  - » Simple Expressions
  - » Lambda Expressions
  - » Binding Variables and Expressions
  - » Recursion
- **Language Levels**
- **Debugging**