Topics Today

- Scheme
  - The Scheme Language
  - Running Scheme Code
    - racket
    - DrRacket

Scheme Built-Ins: car

- The car function return the first element in a list.

- Input: (car '(a b c))
- Output:

Scheme Built-Ins: car

- The car function return the first element in a list.

- Input: (car '(a b c))
- Output: 'a

- Input: (car '((a) (b) (c)))
- Output:

Scheme Built-Ins: car

- The car function return the first element in a list.

- Input: (car '(a b c))
- Output: 'a

- Input: (car '((a) (b) (c)))
- Output: '(a)

- Input: (car (car '((a b) (c d))))
- Output:

Scheme Built-Ins: car

- The car function return the first element in a list.

- Input: (car '(a b c))
- Output: 'a

- Input: (car '((a) (b) (c)))
- Output: '(a)

- Input: (car (car '((a b) (c d))))
- Output: 'a

Scheme Built-Ins: cdr

- The cdr function returns the list without its first element

- Input: (cdr '(a b c))
- Output:

Scheme Built-Ins: cdr

- The cdr function returns the list without its first element

- Input: (cdr '(a b c))
- Output: '(b c)

- Input: (cdr '((a) (b) (c)))
- Output:

Scheme Built-Ins: cdr

- The cdr function returns the list without its first element

- Input: (cdr '(a b c))
- Output: '(b c)

- Input: (cdr '((a) (b) (c)))
- Output: '((b) (c))

- Input: (cdr (car '((a b) (c d))))
- Output:

Scheme Built-Ins: cdr

- The cdr function returns the list without its first element

- Input: (cdr '(a b c))
- Output: '(b c)

- Input: (cdr '((a) (b) (c)))
- Output: '((b) (c))

- Input: (cdr (car '((a b) (c d))))
- Output: '(b)

- Input: (cdr (cdr '((a b) (c d))))
- Output:

Scheme Built-Ins: cdr

- The cdr function returns the list without its first element

- Input: (cdr '(a b c))
- Output: '(b c)

- Input: (cdr '((a) (b) (c)))
- Output: '((b) (c))

- Input: (cdr (car '((a b) (c d))))
- Output: '(b)

- Input: (cdr (cdr '((a b) (c d))))
- Output: '()

Scheme Built-Ins: cons

- The cons function combines two items. The second item is usually a list.

- Input: (cons 'a '(b c))
- Output:

Scheme Built-Ins: cons

- The cons function combines two items. The second item is usually a list.

- Input: (cons 'a '(b c))
- Output: '(a b c)

- Input: (cons '(a) '(b c))
- Output:

Scheme Built-Ins: cons

- The cons function combines two items. The second item is usually a list.

- Input: (cons 'a '(b c))
- Output: '(a b c)

- Input: (cons '(a) '(b c))
- Output: '((a) b c)

- Input: (cons 'a '())
- Output:

Scheme Built-Ins: cons

- The cons function combines two items. The second item is usually a list.

- Input: (cons 'a '(b c))
- Output: '(a b c)

- Input: (cons '(a) '(b c))
- Output: '((a) b c)

- Input: (cons 'a '())
- Output: ‘(a)

- Input: (cons 'a 'b)
- Output:

Scheme Built-Ins: cons

- The cons function combines two items. The second item is usually a list.

- Input: (cons 'a '(b c))
- Output: '(a b c)

- Input: (cons '(a) '(b c))
- Output: '((a) b c)

- Input: (cons 'a '())
- Output: ‘(a)

- Input: (cons 'a 'b)
- Output: '(a . b) ⟵ **improper list**

Scheme Built-Ins: list

- The list function makes a list containing one or more items.

- Input: (list 'a)
- Output:

Scheme Built-Ins: list

- The list function makes a list containing one or more items.

- Input: (list 'a)
- Output: '(a)

- Input: (list '(a) '(b c) 'd)
- Output:

Scheme Built-Ins: list

- The list function makes a list containing one or more items.

- Input: (list 'a)
- Output: '(a)

- Input: (list '(a) '(b c) 'd)
- Output: '((a) (b c) d)

- Input: (list (car '(a b)) (cdr '(a b)))
- Output:

Scheme Built-Ins: list

- The list function makes a list containing one or more items.

- Input: (list 'a)
- Output: '(a)

- Input: (list '(a) '(b c) 'd)
- Output: '((a) (b c) d)

- Input: (list (car '(a b)) (cdr '(a b)))
- Output: '(a (b))

Scheme Built-Ins: cond

- The cond function acts like switch-case code.

- Input: (cond
  ((> 1 2) "entered first case")
  ((> 2 1) "entered second case")
  (#t "entered default case"))
- Output:

Scheme Built-Ins: cond

- The cond function acts like switch-case code.

- Input: (cond
  ((> 1 2) "entered first case")
  ((> 2 1) "entered second case")
  (#t "entered default case"))
- Output: "entered second case"

Scheme Built-Ins: let

- The let function creates an environment with one or more variables.

- Input: (let
  ((a "hello")
  (b (car '("world" "!"))))
  (list a b))
- Output:

Scheme Built-Ins: let

- The let function creates an environment with one or more variables.

- Input: (let
  ((a "hello")
  (b (car '("world" "!"))))
  (list a b))
- Output: '("hello" "world")

Scheme Built-In Commands

- Other built-in functions in Scheme include:
- if
  - E.g.: (if (> a b) "a is greater" "a is not greater")
- null?
  - E.g.: (if (null? a) "a is an empty list" "a is not empty")
- equal?
  - E.g.: (if (equal? a b) "a and b are identical" "a and b are different")
- You can use semicolons for comments.
  - (car a) ;returns the first element in *a*

Scheme – Defining a Function

- Suppose we want a function that returns the smallest number in a list:

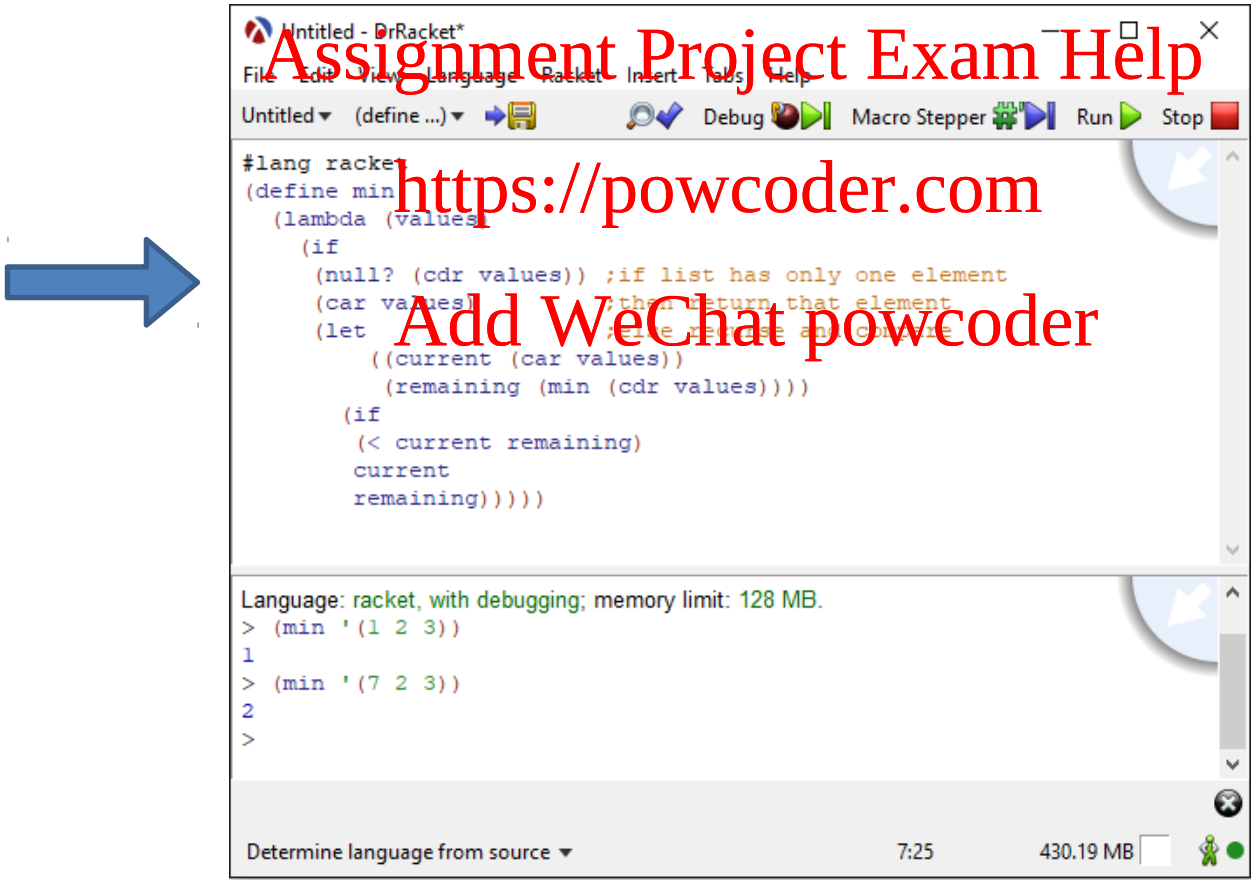

```
(define min
   (lambda (values)
      (if
         (null? (cdr values))     ;if list has only one element
         (car values)             ;then return that element
         (let                     ;else recurse and compare
            ((current (car values))
              (remaining (min (cdr values))))
          (if
             (< current remaining)
             current
             remaining)))))
```

Racket – Running Scheme (via command line)

- Suppose we saved the min function from the last slide to a file: source.rkt
- In racket, we can import this definitions file with the **enter!** Command
- Then we can invoke the min function from this definitions file
- Let's see an example!

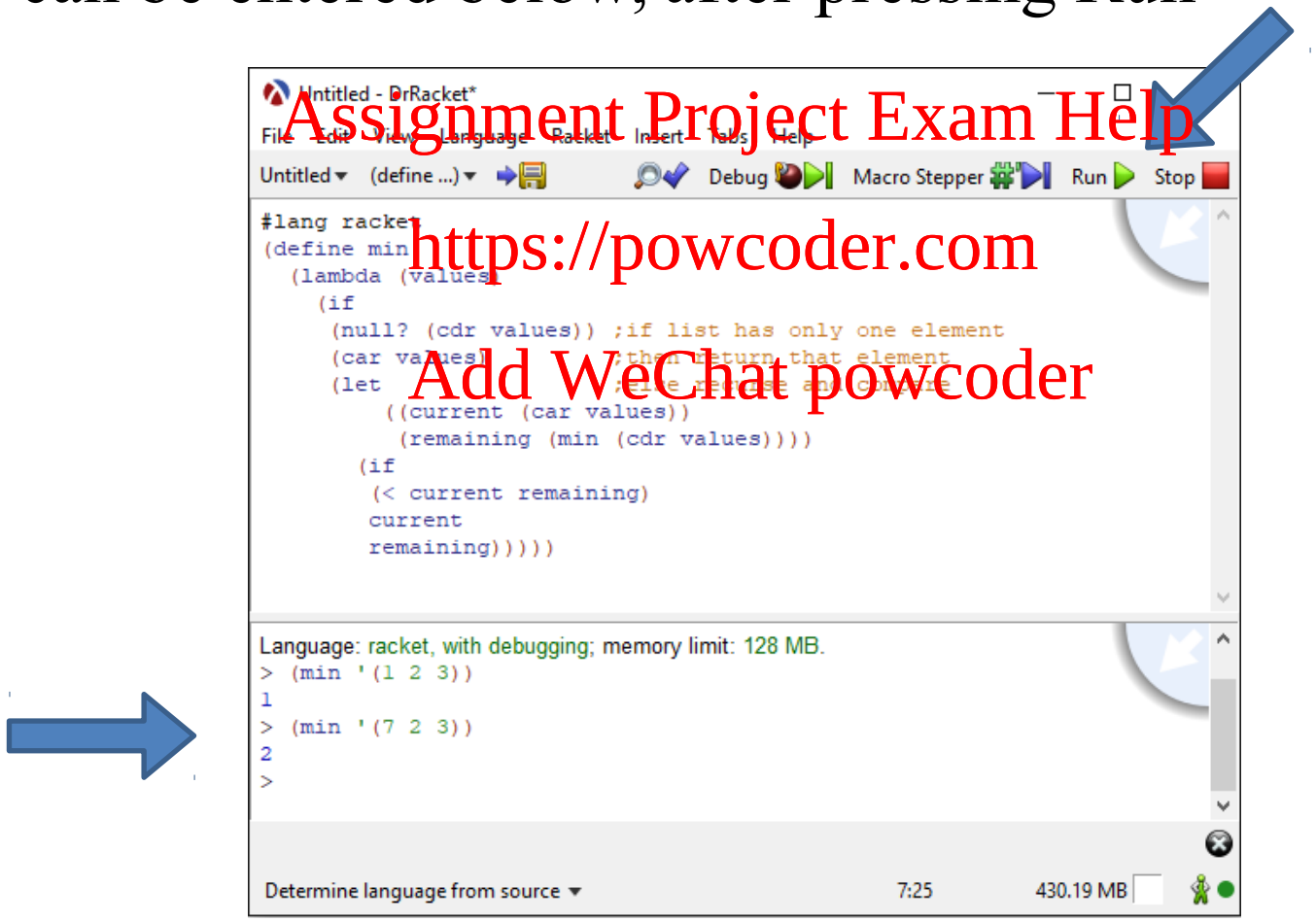
DrRacket – Running Scheme (via GUI)

- Function definitions go in the top text box



DrRacket – Running Scheme (via GUI)

- Commands can be entered below, after pressing Run



Fibonacci function

- Input: n
- Output: the nth number in the Fibonacci sequence

Fibonacci function

- Input: n
- Output: the nth number in the Fibonacci sequence


```
#lang racket
(define (fib n)
  (if
   (or (equal? 1 n) (equal? 2 n)) 1
   (+ (fib (- n 1)) (fib (- n 2)))))
)
```

Median of a sorted list

- Input: a sorted list
- Output:

Median of a sorted list

- Input: a sorted list
- Output: the median number of the list


```
#lang racket
;;Length function
(define (mylength lst)
  (if (null? lst) 0 (+ 1 (mylength (cdr lst))))
)

;;Median of a sorted list
(define (median lst)
  (medianhelp lst 1)
)

(define (medianhelp lst m)
  (cond ((equal? (mylength (cdr lst)) (- m 1)) (car lst))
        ((equal? (mylength (cdr lst)) m) (/ (+ (car lst) (car (cdr lst))) 2))
        (else (medianhelp (cdr lst) (+ m 1)))
  )
)
```

Project 2

- Project 2 will involve implementing something in Scheme.

- It should be released after the mid-term.

- Some tips:

  - Debugging seems more difficult than in C/C++. Start early!
  - Be carefully with your lists; if a solution has incorrect parentheses then it's wrong.
  - I recommend testing your functions for correctness individually.