CSCI 275: Programming Abstractions

Lecture 05: Function Design, Part 1
Spring 2025

Come get your nametag up front as you enter!

Note will be recording audio today!

Stephen Checkoway, Oberlin College
Slides gratefully borrowed from Molly Q Feldman
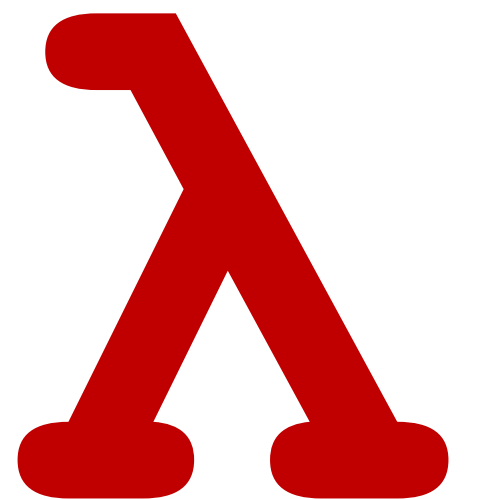
Questions? Concerns?

Goals for Today's Class

Practice, practice, practice

Introduction to some additional helpful constructs for writing procedures in Racket

Functional Language of the Week: OCaml

- Developed by Inria (France)

- One of the core modern variants of the ML language
  - ML is one of the classic functional languages in the same group as Lisp
  - ML handles types in a neat way

- Used as the backend for the theorem proving language Coq

- Jane Street Capital uses OCaml exclusively

Functional Language of the Week: OCaml

```
# let swap_two_elements l =
    match l with
    | fst :: snd :: tl -> snd :: fst :: tl
    | _ -> failwith "Input list must contain at least two elements"
  ;;
```

OCaml

Modules in Racket

Modules in Racket

Each file that starts with `#lang` creates a module named after the file

`#lang` also specifies the language of the file

Racket was designed to implement programming languages
- We will stick mostly with Racket itself
- All of our files start with `#lang racket`

Exposing definitions

`(provide` ...`)`

By default, each definition you make in a Racket file is private to the file

To expose the definition, you use `(provide ...)`

To expose all definitions, you use
`(provide (all-defined-out))`

```
#lang racket
(provide (all-defined-out))

(define mul2
  (lambda (x)
    (* x 2))
```

Exposing only some definitions

(provide sym1 sym2...)

You can specify exactly which definitions are exposed by specifying them via one or more `provide`s

```
#lang racket
(provide foo-a foo-b)
(provide bar-a bar-b)

(define helper ...) ; Not exposed

(define foo-a ...)
(define foo-b ...)

(define bar-a ...)
(define bar-b ...)
```

Importing definitions from modules

(require …**)**

To get access to a module's definitions we need to `require` the module

We see this in the `tests.rkt` files in the assignments require the homework file `(require "hw0.rkt")` imports the definitions from the file `hw0.rkt`

Practice & Function Design

A "complete" program

```
(define sum-positives
 (lambda (lst)
  (cond [(empty? lst) 0]
        [(> (first lst) 0)
         (+ (first lst) (sum-positives (rest lst)))]
        [else (sum-positives (rest lst))])))
```

A "complete" program

This reflects a common pattern: recursion over lists (classic in Racket, all the time!)

```
(define sum-positives
(lambda (lst)
  (cond [(empty? lst) 0]
        [(> (first lst) 0)
         (+ (first lst) (sum-positives (rest lst)))]
        [else (sum-positives (rest lst))])))
```

List functions `empty?`, `first`, `rest`

Base case 0

Recursive calls using the `rest` of the list, combined with the `first` element

Two useful shorthands

1. Racket lets us use λ (cmd-\ or ctl-\ in DrRacket) instead of lambda
   ```
   (define foo
     (λ (x y z)
       (+ x (* y z))))
   ```

2. We can combine define + lambda using a different form of define
   ```
   (define (foo x y z)
     (+ x (* y z)))
   ```

```
(multiply 2 3)  gives 6
(multiply 4 10)  gives 40
What should go in the [    ] ?

(define multiply
  (lambda (n m)
    (cond [(equal? m 0) 0]
          [else [                      ]])))

A. (+ n (multiply n m)

B. (* n (multiply n (- m 1)))

C. (+ n (multiply n (- m 1)))

D. Something else
```

We want to write a procedure `swap` which swaps only the first and second elements of a list. Write swap together with your group!

Tests:

```
(swap '(a b c d)) produces '(b a c d)
(swap '(1 2)) produces '(2 1)
```

Write a procedure `(second-to-last lst)` which returns the second to last element in the list
```
(define (second-to-last lst)
   (cond …))
```

Test your procedure by running it on the lists `'(a b c d)`, `(range 10)`, and `(range 100000)`

`(range n)` returns the list '(0 1 … n-1) so the latter two should return 8 and 99998

Hint: You can use (length lst) to get the length of a list but this is very slow as you'll see if you use it here

Next Up

HW0 due at 11:59pm Friday

Opportunities for help:
- My office hours 1–3 p.m. tomorrow in King 2231