CS 357 - Lab 002

Session 4
let -> lambda

Kage Weiss

CS 357 - Lab 002

TA: Kage Weiss
Office Hours: Email or by appointment.
Contact: mmweiss@unm.edu
Website: **http://cs.unm.edu/~kageweiss/TA/cs357.html** **-- SLIDES POSTED**

- No sign in quiz today, though we've already said several times this will be on the exam, so up to you if you stay
- Today we are prepping for the exam by practicing let -> lambda conversions
- The EXAM is WEDNESDAY the 24th, details on the next slide

EXAM 1

Come prepared to:

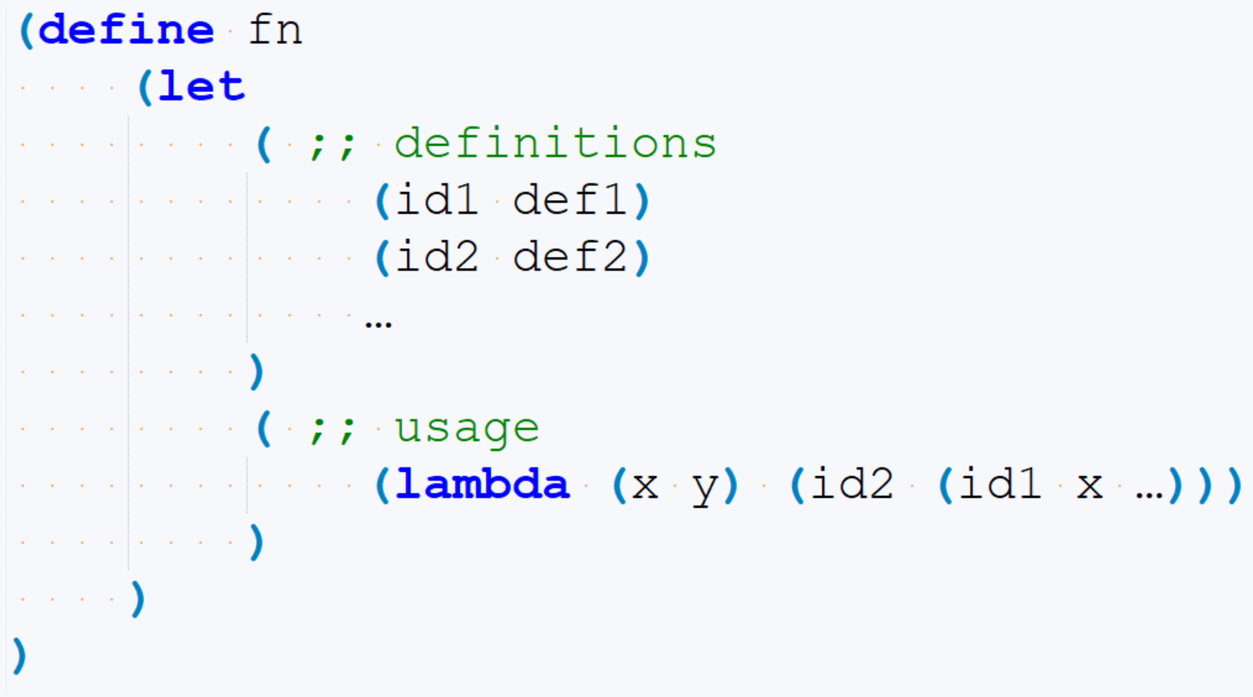
1. Download a skeleton file
2. Open your editor and interpreter/compiler
3. Fill in the skeleton file with your answers
4. Upload your completed skeleton file

- Wednesday Feb. 24th
- Opens ***immediately after lecture***:
  Exam will be available at noon Wed.
- Once you start, you will have 2 hours to submit your exam file.
- A skeleton will be provided, USE IT.
- Exam closes after 24 hours:
  Exam will no longer be available at noon Thurs. the 25th

let, let*, letrec

let

- Allows for scoped definitions
- Does not allow for individual definitions to reference each other
- Does not allow for individual definitions to reference themselves



```
(define fn
    (let
        ( ;; definitions
            (id1 def1)
            (id2 def2)
            …
        )
        ( ;; usage
            (lambda (x y) (id2 (id1 x …)))
        )
    )
)
```

let, let*, letrec

let*

- Allows for scoped definitions
- Allows for individual definitions to reference each other
- Does not allow for individual definitions to reference themselves

```
(define fn
    (let*
        ( ;; definitions
            (id1 def1)
            (id2 (… def2 … id1 …))
            …
        )
        ( ;; usage
            (lambda (x y) (id2 (id1 x …)))
        )
    )
)
```

let, let*, letrec

letrec

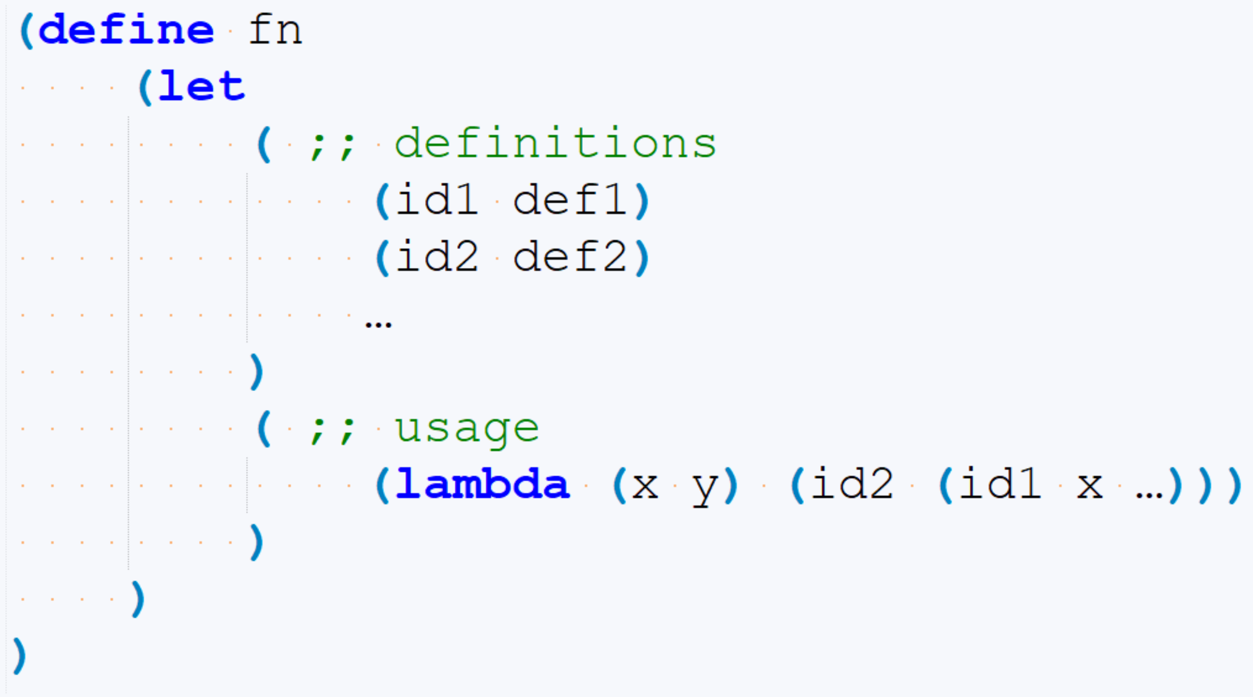
- Allows for scoped definitions
- Allows for individual definitions to reference each other
- Allows for individual definitions to reference themselves

```
(define fn
    (letrec
        ( ;; definitions
            (id1 (… id1 …))
            (id2 (… def2 … def1 …))
            …
        )
        ( ;; usage
            (lambda (x y) (id2 (id1 x …)))
        )
    )
)
```

let, let*, letrec

let

- Allows for scoped definitions
- Does not allow for individual definitions to reference each other
- Does not allow for individual definitions to reference themselves



```
(define fn
    (let
        ( ;; definitions
            (id1 def1)
            (id2 def2)
            …
        )
        ( ;; usage
            (lambda (x y) (id2 (id1 x …)))
        )
    )
)
```

let, let*, letrec

let

- Allows for scoped definitions
- Does not allow for individual definitions to reference each other
- Does not allow for individual definitions to reference themselves

- Definitions are all at the same level
- They cannot reference each other, and so are simply “assigned” to values
- What other notation do we have for naming values for use in code?
  - (hint what is this lecture?)
- It’s Lambda notation

let, let*, letrec

let*

- Allows for scoped definitions
- Allows for individual definitions to reference each other
- Does not allow for individual definitions to reference themselves

```
(define fn
    (let*
        ( ;; definitions
            (id1 def1)
            (id2 (… def2 … id1 …))
            …
        )
        ( ;; usage
            (lambda (x y) (id2 (id1 x …)))
        )
    )
)
```

let, let*, letrec

let*

- Allows for scoped definitions
- Allows for individual definitions to reference each other
- Does not allow for individual definitions to reference themselves

- Now we need to be able to refer to values named higher up the let*
- Not possible if they’re all the same level… How can we solve this?
- Nested Lambda scopes!

let, let*, letrec

letrec

- Allows for scoped definitions
- Allows for individual definitions to reference each other
- Allows for individual definitions to reference themselves

```
(define fn
    (letrec
        ( ;; definitions
            (id1 (… id1 …))
            (id2 (… def2 … def1 …))
            …
        )
        ( ;; usage
            (lambda (x y) (id2 (id1 x …)))
        )
    )
)
```

let, let*, letrec

letrec

- Allows for scoped definitions
- Allows for individual definitions to reference each other
- Allows for individual definitions to reference themselves

- Now they have to be able to refer to themselves
- I’ll leave this one to the book and the internet, but it’s absolutely good practice
- Try to avoid infinite recursion, that’ll come later in Haskell

CS 357 - Lab 002

Go forth,
write your software.

Remember, these slides are available:

cs.unm.edu/~kageweiss/TA/cs357.html