CSE 114A Lecture 3

agenda:
- ✓ Announcements
- ✓ Recap last time
  - lambda calc syntax & substitution rule
  - scope of variables; free & bound variables
  - capture-avoiding substitution
  - renaming
  - Live coding in Elsa!
    - back to booleans / conditionals
    - numbers
    - pairs (if time)
    - recursion (if time)

---

lambda calculus

syntax $e ::= x \mid \lambda x.\ e \mid e_1\ e_2$

- x: variables
- $\lambda x.\ e$: function definition aka lambda abstraction
- $e_1\ e_2$: function application aka function call

semantics

$(\lambda x.\ e_1)\ e_2 \longrightarrow e_1[x := e_2]$ (steps to, =b>)

- $(\lambda x.\ e_1)$: body
- β-substitution
- replace all occurrences of x in e_1 with e_2"

What's a "value"?

$\lambda x.\ e$ ← this is a value!

Even if this e here is some large expression, if the expression can't be evaluated further, we have a value.

$(\lambda x.\ (\lambda y.\ x))\ y \longrightarrow (\lambda y.\ x)[x := y]$

$\lambda y.\ y$

What went wrong here?
y got captured underneath λy

$(\lambda x.\ (\lambda y.\ x))\ y$

we should rename y to something that won't accidentally capture a variable.

$(\lambda x.\ (\lambda z.\ x))\ y$

$\longrightarrow^*$ $\lambda z.\ y$ ← This is what we actually want. We renamed y to z, and then we safely do substitution.

$(\lambda x.\ e_1)\ e_2 \longrightarrow e_1[x := e_2]$

"e_1, but with all free occurrences of x replaced with e_2, except when doing so would capture variables in e_2."

This is called capture-avoiding substitution.

Free and bound variables

a "binder"

$\lambda x.\ x$ — x is bound by λx. No free variables.

$\lambda x.\ (\lambda y.\ x)$ — x is bound by λx. No free variables.

$\lambda x.\ \lambda y.\ z\ y$

y is bound by λy.
z is free.

$\lambda z.\ (\lambda x.\ \lambda y.\ z\ y)$

Now z is no longer free.