

CSE114A Lecture 2

Agenda:

- intro to lambda calculus

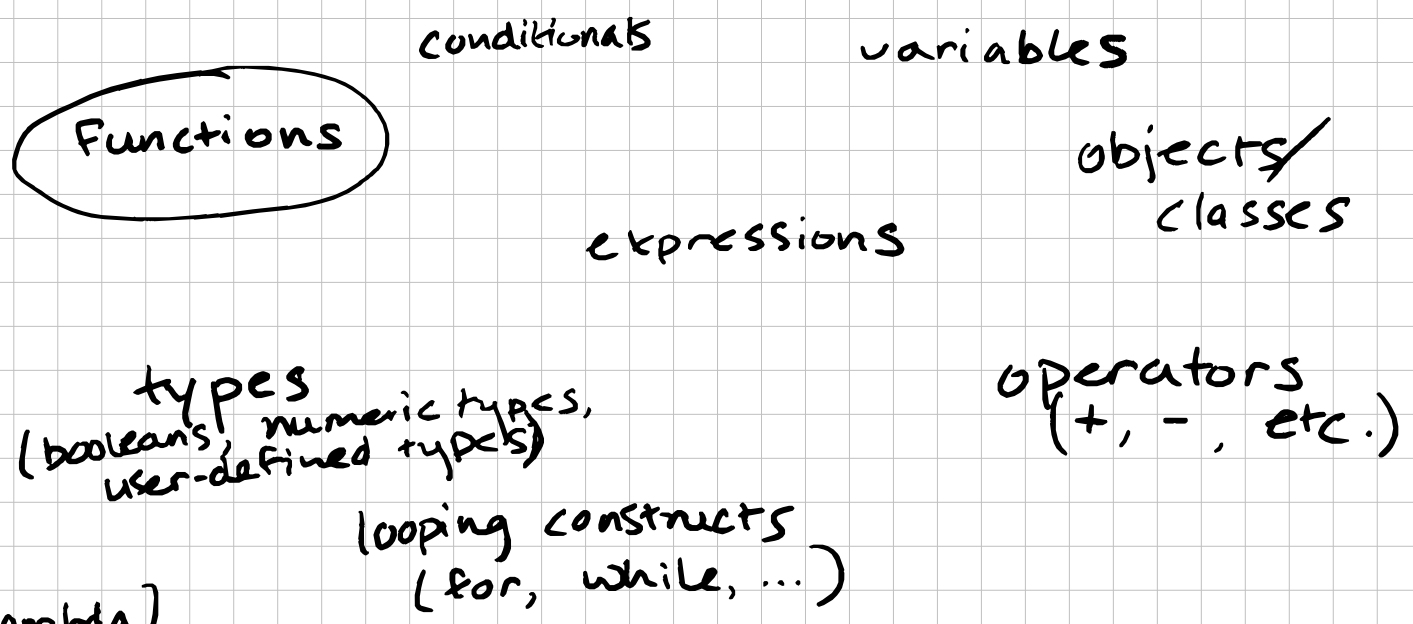
the Greek letter λ a system of reasoning.

1936 - Turing machines (Alan Turing) were invented

1936 - λ -calculus was invented (Alonzo Church)

both universal models of computation.

What features would you expect a PL to have?

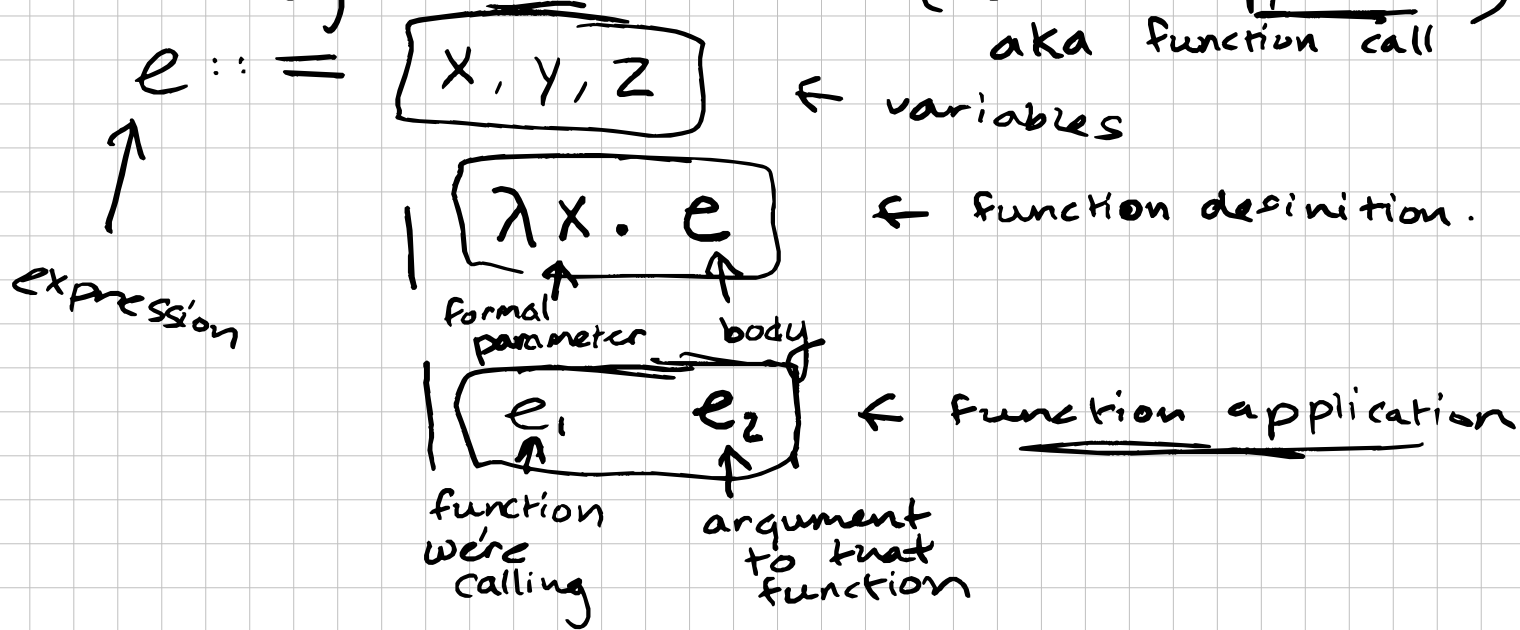


λ -calculus only has functions.

everything else we might want has to be encoded using functions!

- Functions

- way to define them (function definition)
- way to call them (function application) aka function call



✓ Syntax - what programs look like.

Semantics - what programs mean.

$\lambda y. y$ \leftarrow also the identity function

$\lambda x. x$ \leftarrow the identity function.

takes an argument and just returns that argument

`def identity(y):`
`return y` \leftarrow identity function in Python.

$\lambda z. (\lambda x. x)$ \leftarrow a function that returns the identity function

$\lambda y. (\lambda x. x)$ \leftarrow so is this

$\lambda z. (\lambda y. y)$ \leftarrow and so is this

$\lambda f. f (\lambda x. x)$ \leftarrow a function that applies its argument to the identity function
an application!

paper syntax

Elsa syntax

$\lambda x. x$

$\backslash x \rightarrow x$

$\lambda f. f (\lambda x. x)$

$\backslash f \rightarrow f (\backslash x \rightarrow x)$

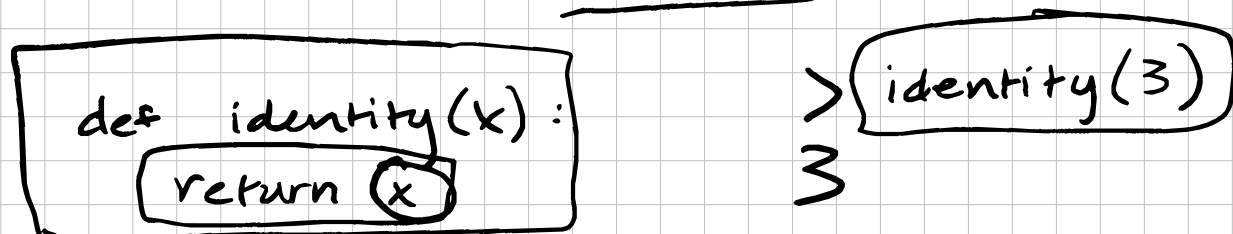
What about semantics?

We've talked about Syntax, but what do λ -calculus programs mean?

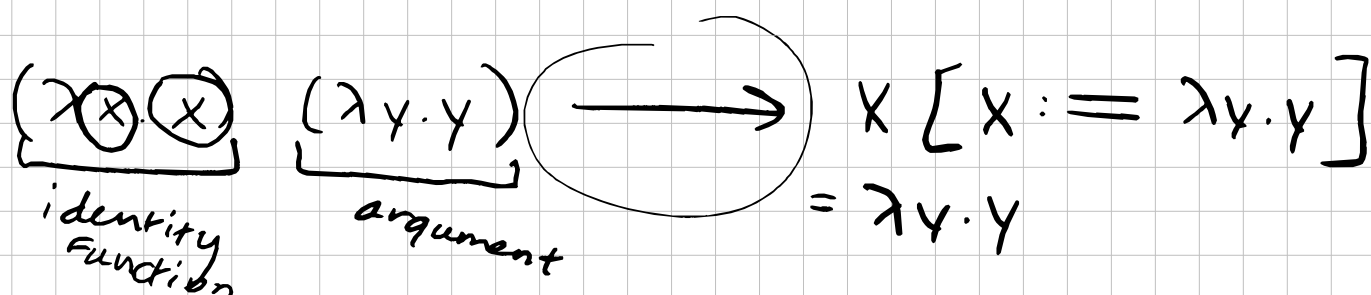
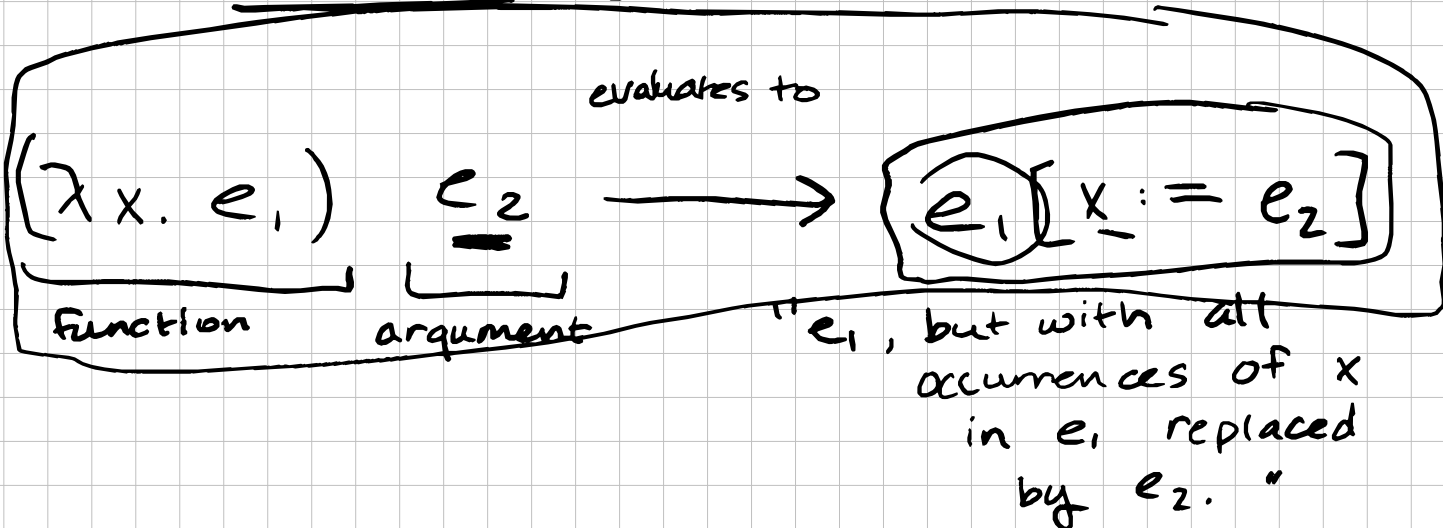
One way to talk about meaning is operational semantics:

how do programs execute step by step?

The essence of computation in λ -calculus is substitution.



We substituted 3 for x.

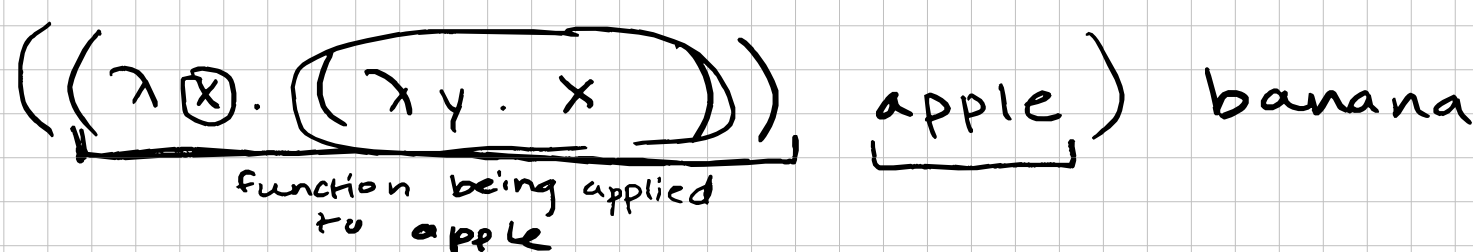


Let's try encoding Booleans and conditionals using λ -calculus.

but first... how do we write functions with more than one argument?

$\lambda x y. x$ ← what we want

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



$(\lambda y. \underline{\text{apple}})$ banana

apple

Instead of a function that takes multiple arguments, we have:
a function that takes the 1st argument and returns

a function that takes the 2nd argument

(...and returns

a function that takes the 3rd argument, etc.)

let TRUE = $\lambda x. (\lambda y. x)$

let FALSE = $\lambda x. (\lambda y. y)$

(A classic way to encode Booleans in lambda calculus.)

Known as a Church encoding.

if <conditionals> then <branch 1> else <branch 2>

let ITE = $\lambda b. (\lambda x. (\lambda y. (b x) y))$

↑
"if then else"

How to read this:

$(b x) y$
Apply b to x ,
then apply the
result of that
to y .