

CSE114A lecture 2

lambda calculus!

↑
the Greek
letter λ

↑
a system of
reasoning

1936 - both Turing machines
and λ -calculus
were invented!

A very simple programming language

variables

~~types~~

~~loops~~

recursion
(encodable)

~~state
assignment~~

functions
abstractions

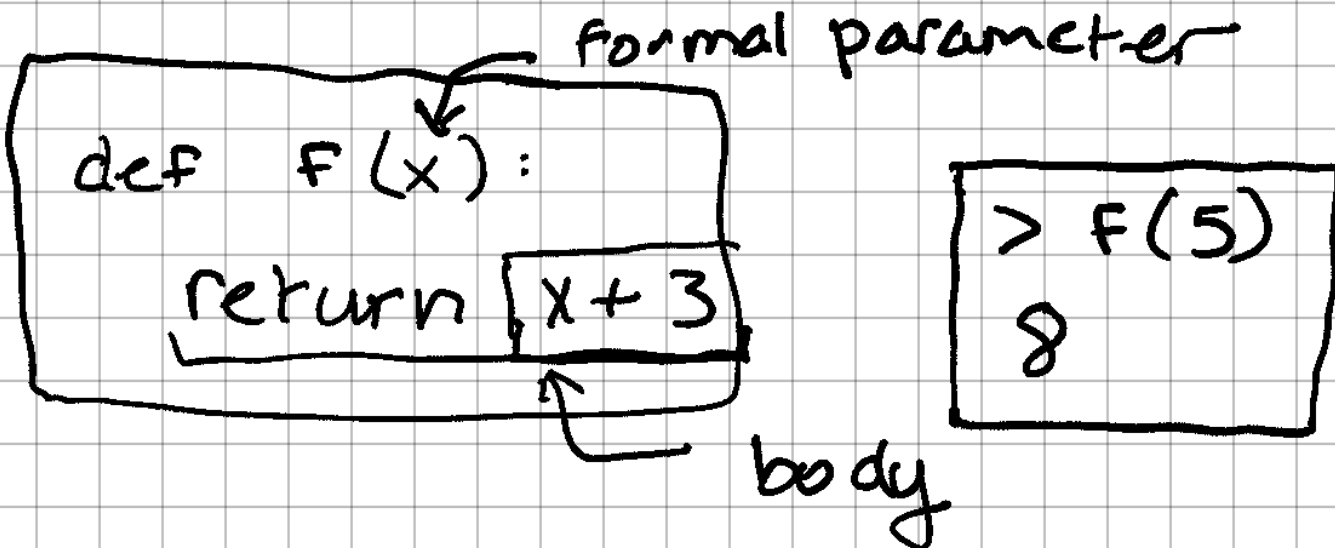
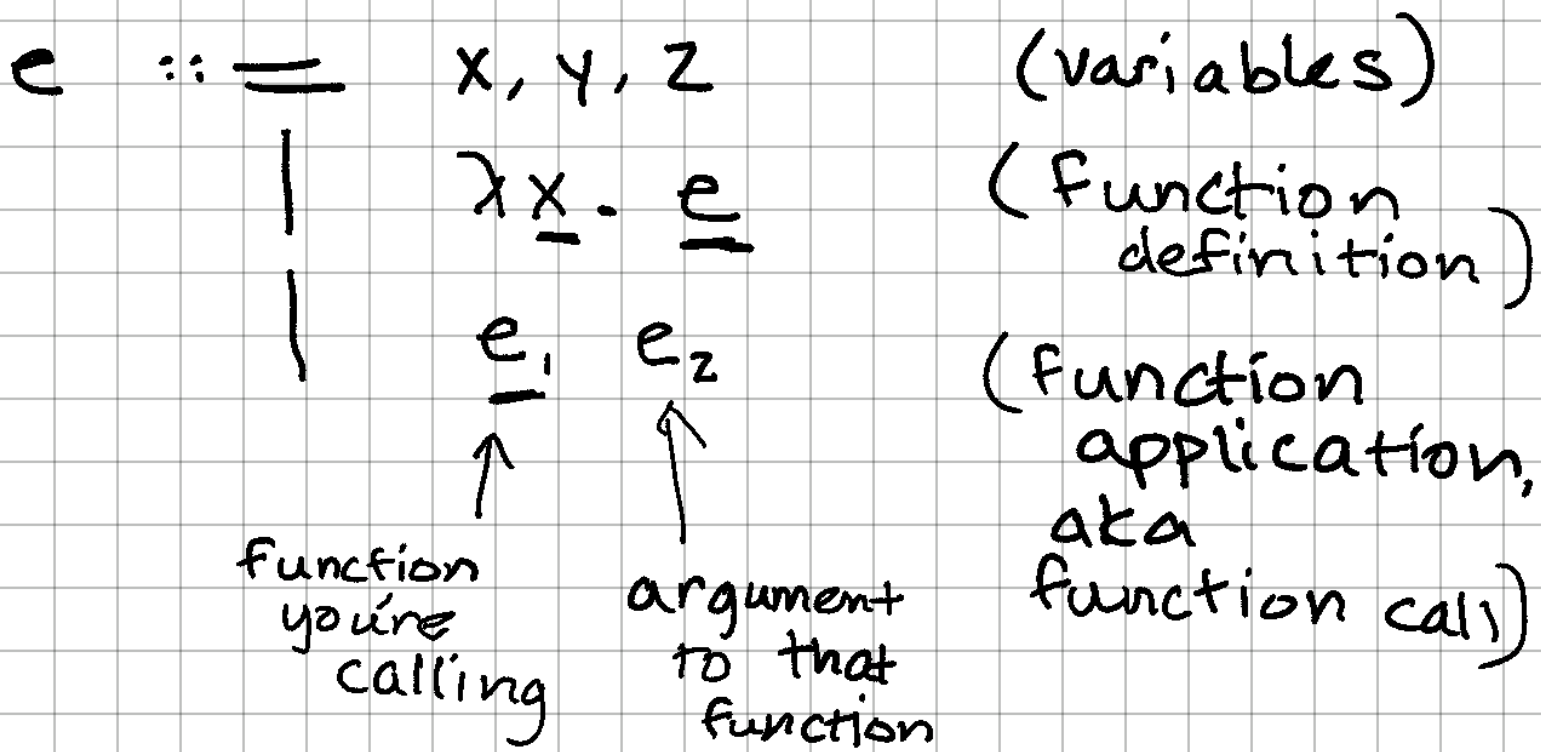
operators
(encodable)

~~memory model~~

conditionals
(encodable)

All we've got:

- variables
- functions
 - way to define them
 - way to call them



Computation in λ -calculus
all boils down to
substitution.

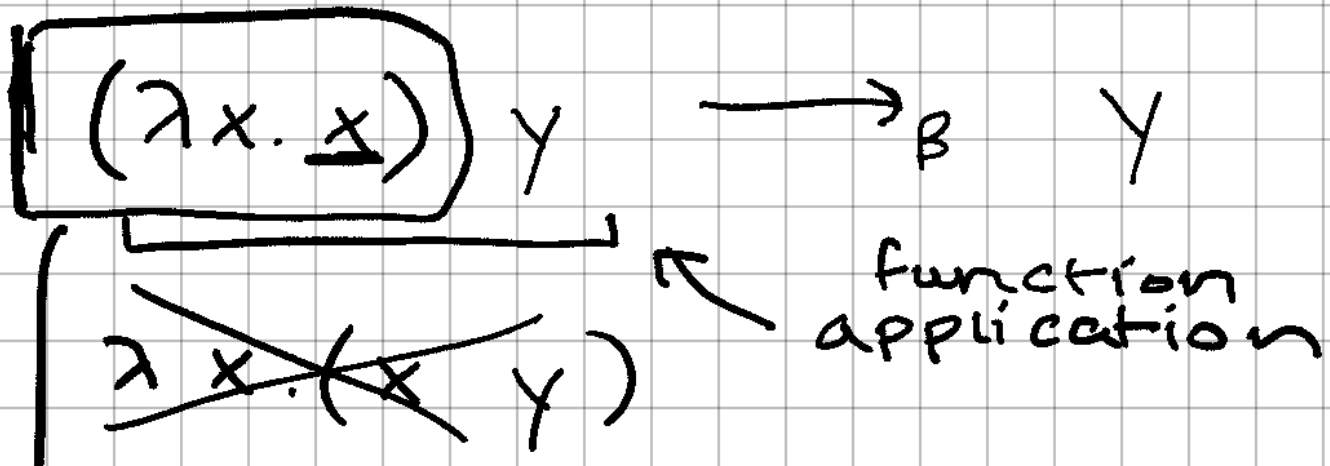
The simplest function:
the identity function

$\lambda x. x$

$\lambda x \rightarrow x$

(on-paper syntax)

(Elsa syntax)



```
def f(x):  
    return x
```

```
> f(z)  
z
```

The essence of computation
in λ -calculus is
substitution, via
 β -reduction

$$\underbrace{(\lambda \underline{x}. \overbrace{e_1}^{\text{body}})}_{\text{Function}} \underbrace{e_2}_{\text{argument}} \longrightarrow_{\beta} \underbrace{e_1}_{\text{body}} [\underline{x} := e_2]$$

" e_1 , but with all*
occurrences* of x replaced
with e_2 "

* note:
this
is
a lie

$$(\lambda x. \underbrace{x}_{\text{argument}}) (\lambda y. \underbrace{y}_{\text{argument}}) \longrightarrow_{\beta} \lambda y. y$$

$$\underbrace{x [x := \lambda y. y]}_{\text{body}} = \underline{\lambda y. y}$$

$\left. \begin{array}{l} \lambda x. x \\ \lambda y. y \end{array} \right\}$ these expressions
are α -equivalent

Cool thing:

λ -calculus is a naturally parallelizable model of computation

(because you can pick anywhere in an expression to evaluate)

Let's program with λ -calc!

let TRUE = $\lambda x y. x$

let FALSE = $\lambda x y. y$

How do we encode 'if'?

let ITE = $\lambda b x y. \boxed{b x y}$

condition you're checking

value if b evals to true

value if b evals to false

ITE TRUE FALSE TRUE

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

let TRUE = $\lambda x y. x$

let FALSE = $\lambda x y. y$

recall: $(\lambda x. \underline{e_1}) e_2 \rightarrow_B e_1[x := e_2]$

$(\lambda \underline{b} x y. \underline{b x x}) \underline{\text{TRUE}} \text{FALSE TRUE}$
 \rightarrow_B

$(\lambda x y. \text{TRUE } x y) \text{FALSE TRUE}$
 \rightarrow_B

$(\lambda y. \text{TRUE FALSE } y) \text{TRUE}$
 \rightarrow_B

TRUE FALSE TRUE
 \rightarrow_{def}

$(\lambda \underline{x} y. x) \text{FALSE TRUE}$
 \rightarrow_B

$(\lambda y. \text{FALSE}) \text{TRUE}$

\rightarrow_B

FALSE

$$\underbrace{(\lambda x. x)}_{\text{function}} \underbrace{y}_{\text{arg}} \rightarrow_{\beta} y$$

$$\overline{(\lambda \underline{x}. (\lambda \underline{y}. \underline{x}))} \quad \textcircled{y}$$

\rightarrow_{β} in a naive way:

$$\lambda y. y$$

This is wrong! $\ddot{\smile}$

To solve this problem,
we need capture-avoiding
substitution.

$$\left. \begin{array}{l} \lambda \underline{x}. \underline{x} \\ \lambda \underline{y}. \underline{y} \end{array} \right\} \alpha\text{-equivalent}$$

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

\rightarrow_{α}

$$(\lambda \underline{x}. (\lambda \underline{z}. \underline{x})) y$$

\rightarrow_{β}

$$\lambda z. y$$

We replaced all free occurrences
of the bound variable in the
body.