Lambdas
CIS352 — Fall 2022
Kris Micinski
First-Class Functions

• In Racket, functions are **first-class** values
• Can be bound to vars, returned from fns, etc..
• Languages w/ functions as values are **functional**
Lambdas (in Racket)

- \((\text{lambda} \ (x_0 \ x_1 \ \ldots) \ \text{body})\)
  - Anonymous function: bind \(x_0, \ldots\) in body
  - Can appear at any callsite (just like an identifier)

\[
\begin{align*}
\text{(define f (lambda (x) x))} \\
\text{(define (double g)} \\
\quad \text{(lambda (x) (g (g x)))))}
\end{align*}
\]
(define f (lambda (x) x))
(define (double g)
  (lambda (x) (g (g x)))))

Evaluate the following expressions:
• (f 1)
• ((double f) 42)
• ((double (lambda (x) (* x 2))) 2)
Exercise

Write a function, (foo f), that:
• Accepts a function f, maps ints to ints
• ((foo f) x) = (f |x|), |x| is abs. value of x
Textual Reduction of Lambdas

- Previously, we assumed environment of definitions
- Instead, can think of lambdas as primitive
- Environment maps identifiers to lambdas

(define (f x) x)
;; equiv
(define f (lambda (x) x))
Textual Reduction of Lambdas

- After reducing all args to values, substitute (into the body) the actual arguments in place of the formal arguments.

```
((lambda (x y) x) (+ 1 1) 3)
=> ((lambda (x y) x) 2 3)
=> 2
```
Exercise

Use textual reduction to reduce the following:

```
((((lambda (x) x) (lambda (x) x))
  ((lambda (x) x) (lambda (x) x)))
 (+ 1 2))
```

Hint: remember, in **applicative order** we always evaluate the **leftmost, innermost** application. In other words, we process `(e0 e1 ...)` by reducing `e0 ...` to values in order, then applying.
Use textual reduction to reduce the following:

\[
(((\lambda (x) \ x) \ (\lambda (x) \ x))
 ((\lambda (x) \ x) \ (\lambda (x) \ x)))
 (+ \ 1 \ 2))
\]

If this sounds complicated, you would be right to just think about it as “left to right”
Languages w/o First-Class Functions

• In modern times, somewhat hard to imagine
• C is a good example: procedural but **not** functional
• C callsites: quasi-functional behavior via fn pointers
  • But not really: C doesn’t have **closures**

```c
// The C library QuickSort function
void qsort(void *base, // array to sort
           int items, // really size_t
           int elem_size,
           // pointer to compare fn
           int (*compare)(void*, void*))
```