First-Class Functions in Racket

Design of Programming Languages

Modified Slides of Lyn Turbak

First-Class Values

A value is **first-class** if it satisfies all of these properties:

- It can be named by a variable
- It can be passed as an argument to a function;
- It can be returned as the result of a function;
- It can be stored as an element in a data structure (e.g., a list);
- It can be created in any context.

Examples from Racket: numbers, boolean, strings, characters, lists, … and **functions**!

Functions can be Named

```
(define dbl (λ (x) (* 2 x)))

(define avg (λ (a b) (/ (+ a b) 2))))

(define pow
  (λ (base expt)
    (if (= expt 0)
        1             ⇒* 3
        (* base (pow base (- expt 1))))))
```

Recall syntactic sugar:

```
(define (dbl x) (* 2 x))

(define (avg a b) (/ (+ a b) 2)))

(define (pow base expt) …)
```

Functions can be Passed as Arguments

```
(define app-3-5 (λ (f) (f 3 5))
(define sub2 (λ (x y) (- x y)))


({app-3-5} sub2)
⇒((λ (f) (f 3 5)) {sub2}) [varref]
⇒{((λ (f) (f 3 5)) (λ (x y) (- x y)))} [varref]
⇒{((λ (x y) (- x y)) 3 5)} [function call]
⇒{(- 3 5)} [function call]
⇒-2 [subtraction]
```

More Functions-as-Arguments

it's your turn

What are the values of the following?

```
(app-3-5 avg)


(app-3-5 pow)


(app-3-5 (λ (a b) a))


(app-3-5 +)
```

Functions can be Returned as Results from Other Functions

```
(define make-linear-function
  (λ (a b) ; a and b are numbers
    (λ (x) (+ (* a x) b))))

(define 4x+7 (make-linear-function 4 7))

(4x+7 0)

(4x+7 1)

(4x+7 2)

(make-linear-function 6 1)

((make-linear-function 6 1) 2)

((app-3-5 make-linear-function) 2)
```

More Functions-as-Returned-Values

it's your turn

```
(define flip2
  (λ (binop)
    (λ (x y) (binop y x))))

((flip2 sub2) 4 7)

(app-3-5 (flip2 sub2))

((flip2 pow) 2 3))

(app-3-5 (flip2 pow))

(define g ((flip2 make-linear-function) 4 7))

(list (g 0) (g 1) (g 2))

((app-3-5 (flip2 make-linear-function)) 2)
```

Functions can be Stored in Lists

it's your turn

```
(define funs (list sub2 avg pow app-3-5
                   make-linear-function flip2))

((first funs) 4 7)

((fourth funs) (third funs))

((fourth funs) ((sixth funs) (third funs)))

(((fourth funs) (fifth funs)) 2)

(((fourth funs) ((sixth funs) (fifth funs))) 2)
```

Functions can be Created in Any Context

- In some languages (e.g., C) functions can be defined only at top-level. One function cannot be declared inside of another.

- Racket functions like `make-linear-function` and `flip2` depend crucially on the ability to create one function inside of another function.

Python Functions are First-Class!

```
def sub2 (x,y):
  return x - y

def app_3_5 (f):
  return f(3,5)
```

```
def make_linear_function(a, b):
  return lambda x: a*x + b

def flip2 (binop):
  return lambda x,y: binop(y,x)
```

```
In [2]: app_3_5(sub2)
Out[2]: -2

In [3]: app_3_5(flip2(sub2))
Out[3]: 2

In [4]: app_3_5(make_linear_function)(2)
Out[4]: 11

In [5]: app_3_5(flip2(make_linear_function))(2)
Out[5]: 13
```

JavaScript Functions are First-Class!

```
function sub2 (x,y){
{ return x-y; }

function app_3_5 (f)
{ return f(3,5); }
```

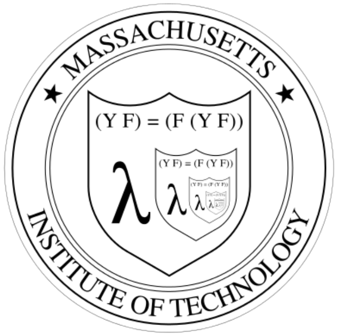
```
function make_linear_function(a,b) {
  return function(x) {return a*x + b;};
}

function flip2(binop) {
  return function(x,y)
    { return binop(y,x); }
}
```

```
> app_3_5(sub2)
< -2
> app_3_5(flip2(sub2))
< 2

> app_3_5(make_linear_function)(2)
< 11

> app_3_5(flip2(make_linear_function))(2)
< 13
```

Summary (and Preview!)

Data and procedures and the values they amass,
Higher-order functions to combine and mix and match,
Objects with their local state, the messages they pass,
A property, a package, a control point for a catch —
In the Lambda Order they are all first-class.
One Thing to name them all, One Thing to define them,
One Thing to place them in environments and bind them,
In the Lambda Order they are all first-class.

Abstract for the *Revised4 Report on the Algorithmic Language Scheme (R4RS)*, MIT Artificial Intelligence Lab Memo 848b, November 1991



Emblem for the Grand Recursive Order
of the Knights of the Lambda Calculus

More Functions-as-Arguments Solutions

it's your turn

What are the values of the following?

```
(app-3-5 avg) ⇒* 4

(app-3-5 pow) ⇒* 243 ; 3^5

(app-3-5 (λ (a b) a)) ⇒* 3

(app-3-5 +) ⇒* 8
```

Functions can be Returned as Results from Other Functions Solutions

it's your turn

```
(define make-linear-function
   (λ (a b) ; a and b are numbers
     (λ (x) (+ (* a x) b))))
make-linear-function ↦ (λ (a b) (λ (x) (+ (* a x) b))) )

(define 4x+7 (make-linear-function 4 7))
4x+7 ↦ (λ (x) (+ (* 4 x) 7)))
```

; Note: This illustrates that functions are data structures! **make-linear-function**
; returns something similar to a Java object that "remembers" instance vars a and b!

```
(4x+7 0) ⇒* 7

(4x+7 1) ⇒* 11

(4x+7 2) ⇒* 15

(make-linear-function 6 1) ⇒* (λ (x) (+ (* 6 x) 1)))

((make-linear-function 6 1) 2)⇒* 13

((app-3-5 make-linear-function) 2)⇒* 11
```

More Functions-as-Returned-Values Solutions

it's your turn

```
(define flip2
  (λ (binop)
    (λ (x y) (binop y x))))
flip2 ↦ (λ (binop) (λ (x y) (binop x y)) )

((flip2 sub2) 4 7)⇒* 3

(app-3-5 (flip2 sub2))⇒* 2

((flip2 pow) 2 3)) ⇒* 9 ; 3^2

(app-3-5 (flip2 pow)) ⇒* 125 ; 5^3

(define g ((flip2 make-linear-function) 4 7))
g ↦ (λ (x) (+ (* 7 x) 4)))

(list (g 0) (g 1) (g 2)) ⇒* '(4 11 18)

((app-3-5 (flip2 make-linear-function)) 2) ⇒* 13
```

Functions can be Stored in Lists Solutions

```
(define funs (list sub2 avg pow app-3-5
                   make-linear-function flip2))
```

; funs is a list of 6 functions. In Racket, the printed representation of this list is:

```
'(#<procedure:sub2> #<procedure:avg>
  #<procedure:pow> #<procedure:app-3-5>
  #<procedure:make-linear-function> #<procedure:flip2>)
```

```
((first funs) 4 7) ⇒* -3

((fourth funs) (third funs))⇒* 243 ; 3^5

((fourth funs) ((sixth funs) (third funs)))⇒* 125 ; 5^3

(((fourth funs) (fifth funs)) 2) ⇒* 11

(((fourth funs) ((sixth funs) (fifth funs))) 2) ⇒* 13
```