CS61A Lecture 3

2011-06-22

Colleen Lewis

REVIEW: Defining functions

```
(define (square x) (* x x))
```

- Define
- Square to be a procedure
- That takes in an argument x
- And multiplies x by x

A new way for defining functions! λ

Procedures are "things"

```
STk> +
#[closure arglist=args 196d20]
STk> (define (square x) (* x x))
square
STk> square
#[closure arglist=(x) 7ff09c08]
STk> (define square _______________
```

Defining a procedure isn't that different than defining a variable

A new way for defining functions! λ

Special form

```
STk>(define y 100)
y
STk>(define sq (lambda(x)(* x x)))
sq
STk> sq
#[closure arglist=(x) 7ff0c248]
STk> (sq 6)
36
```

Lambda returns a procedure

A procedure

```
(define sq(lambda(x)(* x x))
```

- Define
- sq to be ~~a procedure~~
- That takes in an argument x
- And multiplies x by x

Defining procedures using lambda λ

```
(define (sum a b c)
  (lambda(a b c)
    (+ a b c)))
```

This version without lambda is called "syntactic-sugar"

Try it! Rewrite using λ

```
(define average x y)
  (lambda(x y)
    (/(+ x y) 2))
```

A)easy B)medium C)hard D)stuck

Try It!

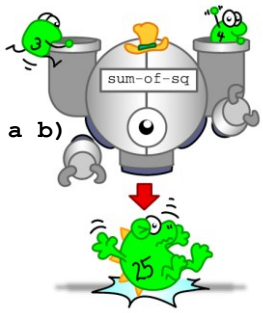
```
(define addTwo (lambda(y)(+ y 2)))
(define (addTwo y) (+ y 2))
```

Step 1: Rewrite `addTwo` with syntactic sugar!

Step 2: Vote: How would you call `addTwo`?

A) ((addTwo 3))
B) (addTwo 3) Correct Answer
C) ((addTwo) 3)
D) '(addTwo 3)
E) No clue!

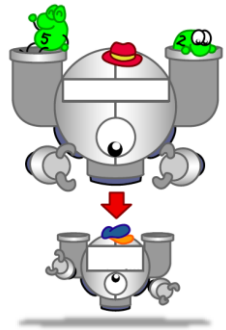
"Normal functions"




```
(define (sum-of-sq a b)
    (+
      (* a a)
      (* b b)))
```

CS Illustrated .berkeley.edu

Functions can return functions!



CS Illustrated .berkeley.edu

Rewriting using syntactic sugar

```
(define add-punctuation
    (lambda (punctuation)
      (lambda (sent)
        (se sent punctuation))))

(define (add-punctuation punctuation)
  (lambda (sent)
    (se sent punctuation)))
```

```
(define add-punctuation
    (lambda (punctuation)
      (lambda (sent)
        (se sent punctuation))))
```

These are equivalent!

```
(define (add-punctuation punctuation)
    (lambda (sent)
      (se sent punctuation)))
```

```
I) add-punctuation
II)(add-punctuation '?)
```

A) Neither are functions
B) Only `I` is a function
C) Only `II` is a function
Correct Answer D) `I`&`II` are both functions
E) Not sure

Calling a procedure that returns a procedure

```
(define (add-punctuation punctuation)
    (lambda (sent)
        (se sent punctuation)))

(define add-exclamation
        (add-punctuation '!))

(define add-question-mark
        (add-punctuation '?))
```

Calling a procedure that returns a procedure

```
(define (add-punctuation punctuation)
    (lambda (sent)
        (se sent punctuation)))

(define add-exclamation
        (add-punctuation '!))

>(add-exclamation '(the potluck will be awesome))
(the potluck will be awesome !)
```

```
(define (a b)
     (lambda (c d)
        (lambda (e f)
            '(hello))))

STk> ____ a _____________________
```

How many open parentheses should go before `a` to get the sentence `(hello)` returned?

A) 1 B) 2 C) 3 D)4 E)??

Try to fill in the blanks on both side!

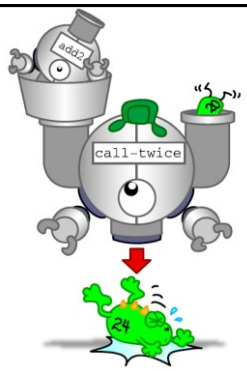
Solution:

```
(define (a b)
     (lambda (c d)
        (lambda (e f)
            '(hello))))
(a 1)
((a 1) 2 3)
(((a 1) 2 3) 4 5)
```

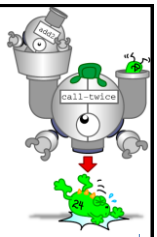
Returns a procedure that takes args `c` & `d`

Returns a procedure that takes args `e` & `f`

Functions can be data to another function!



Functions as Data!




```
(define (addTwo n) (+ n 2))
(define (addFour n)
    (addTwo (addTwo n))

(define (call-twice funct x))
    (funct (funct x)))

STk>(call-twice addTwo 20)
24
```

When can I use different variables?

- A "global" variable, similarly, can be used anywhere:

```
STk>(define pi 3.1415926535)
STk>(define (cat) '(meow meow meow))
```

- Arguments to procedures can be used inside that procedure

```
STk> (define (square x) (* x x))
```

Let (create new variables in definitions)

```
(let
   ( (variable1 value1) ;;definition 1
     (variable2 value2) ;;definition 2
   )
   statement1 ;;body
   statement2 ... )
```

Using `let` to define temporary variables

- `let` lets you define variables within a procedure:

```
(define (scramble-523 wd)
   (let ((second (first (bf wd)))
         (third  (first (bf (bf wd))))
         (fifth  (item 5 wd)))
     (word fifth second third) ) )

(scramble-523 'meaty) ➔ yea
```

Unix Review

- ls _____
- cd folder1 _____
- cd .. _____
- cd _____
- mkdir folder2 _____
- rm file1 _____
- emacs file1 & _____

Try It! SOLUTION

```
(define addTwo (lambda(y)(+ y 2)))
(define (addTwo y) (+ y 2))
```

Step 1: Rewrite `addTwo` with syntactic sugar!

Step 2: Vote: How would you call `addTwo`?

A) ((addTwo 3))
B) (addTwo 3) Correct Answer
C) ((addTwo) 3)
D) '(addTwo 3)
E) No clue!

Unix Review

- ls List contents of folder
- cd folder1 Double click on folder
- cd .. Go UP one folder level
- cd Go to home/main folder
- mkdir folder2 Create new folder
- rm file1 Remove something
- emacs file1 & Create file in current folder