CS61A Lecture 5

2011-06-27

Colleen Lewis

How many ways to make 50 cents?
`first-denomination` Solution

```
(define (first-denomination kinds-of-coins)
  (cond ((= kinds-of-coins 1) 1)
        ((= kinds-of-coins 2) 5)
        ((= kinds-of-coins 3) 10)
        ((= kinds-of-coins 4) 25)
        ((= kinds-of-coins 5) 50)))
STk> (first-denomination 5)
50
STk> (first-denomination 3)
10
STk> (first-denomination 1)
1
```

`count-change`

```
(define (count-change amount)
    (cc  amount  5))

(define (cc amount kinds-of-coins)

    ...

        (first-denomination kinds-of-coins)

    ...
```

`cc` base cases

```
(define (count-change amount)
    (cc  amount  5))
(define (cc amount kinds-of-coins)
STk> (cc _______ 5)
1
STk> (cc _______ 5)
0
STk> (cc 20 _______)
0
```

How many have you figured out?

A) 0
B) 1
C) 2
D) 3

`count-change` base cases

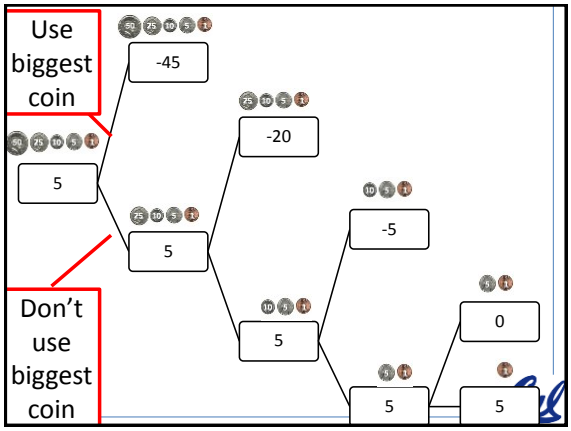
```
(define (cc amount kinds-of-coins)
    (cond ((= amount 0) 1)
          ((< amount 0) 0)
          ((= kinds-of-coins 0) 0)
          (else

((or (< amount 0)(= kinds-of-coins 0)) 0)
```

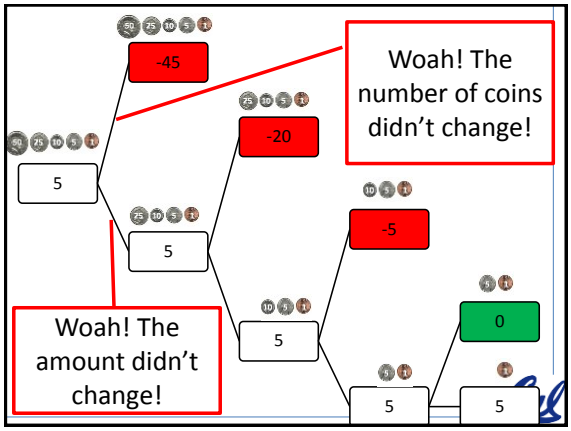
Designing the recursion

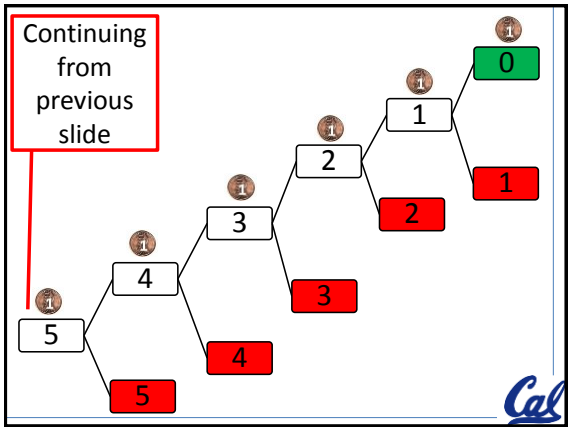
Delegate (make a recursive call):

- Assuming you use the biggest coin (w/ highest value)
- Assuming you don't use the biggest coin

Example: How many ways can you make 5 cents?







`count-change` recursive cases

Blank 1 Blank 2

```
(define (cc amount kinds-of-coins)
…
(+
    (cc
          _____
          _____
    (cc
          _____
          _____
```

Reading Code (Like analyzing a poem)

- Start with simple parts
  - *e.g.* (`first-denomination`)
- Read, re-read, re-read, re-read, test, re-read
  - Re-read specific sections
  - Highlight things you think are important
  - Write down your hypotheses after each reading
    - Tests your hypothesis when possible
  - You're not going to understand it the first or third time you read it!

Same is true for project specs!!!!

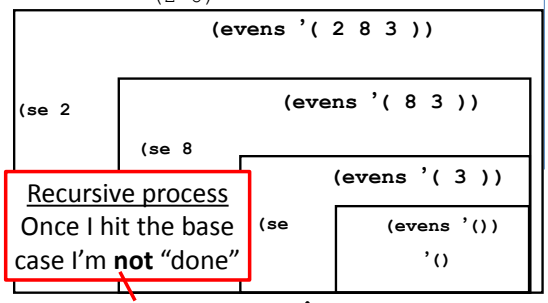
Iterative versus Recursive

Process

(all of which use recursion!)

Remove all non-even numbers from a sentence

```
STk>(evens '(2 8 3))
(2 8)
```



Recursive process
Once I hit the base case I'm **not** "done"

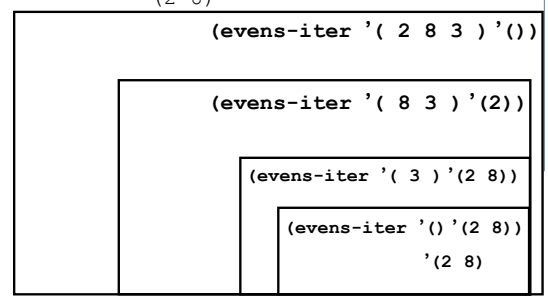
evens

```
(define (evens sent)
    (cond
        ((empty? sent) '())
        ((even? (first sent))
            (se (first sent)
                (evens (bf sent))))
        (else
            (evens (bf sent)))))
```

Waiting to do se

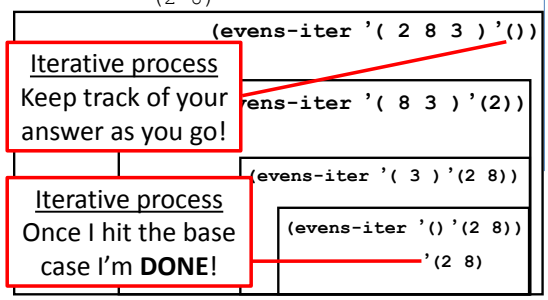
Remove all non-even numbers from a sentence

```
STk>(evens2 '(2 8 3) )
(2 8)
```



Remove all non-even numbers from a sentence

```
STk>(evens2 '(2 8 3) )
(2 8)
```



Iterative process
Keep track of your answer as you go!

Iterative process
Once I hit the base case I'm **DONE**!

evens2

```
(define (evens2 sent)
  (define (evens-iter sent answer)
    (if (empty? sent)
        [          ]
        (evens-iter
            (bf sent)
            [                    ])))
  (evens sent [ ]))
```

Do you like this version better? A) Yes B) No

Factorial

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

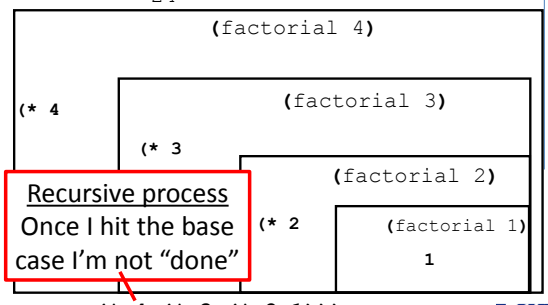
Does this generate:

A) A recursive process

B) An iterative process

Remove all non-even numbers from a sentence

```
STk>(factorial 4)
24
```



Recursive process
Once I hit the base case I'm not "done"

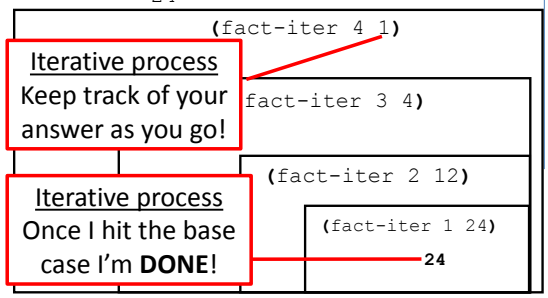
factorial version 2

```
(define (factorial n)
  (define (fact-iter n answer)
    (if (= n 1)
        [          ]
        (fact-iter (- n 1) [          ])))
  (fact-iter n [ ]))
```

Do you like this version better? A) Yes B) No

Remove all non-even numbers from a sentence

```
STk>(factorial 4)
24
```

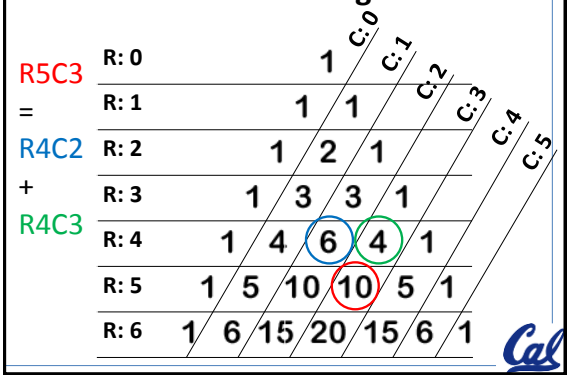


Iterative process
Keep track of your answer as you go!

Iterative process
Once I hit the base case I'm **DONE**!

Pascal's Triangle

R5C3 = R4C2 + R4C3



Recursion (Cont.)

R5C3 = R4C2 + R4C3

(R,C) = (R-1,C-1) + (R-1, C)

```
(define (pascal row col)
  (cond
    ((= col 0) 1)
    ((= col row) 1)
    (else (+ [          ]
```

Does this generate:

A) A recursive process

B) An iterative process

Summary

- `count-change` an example of:
  - Practicing learning to read and trace code
  - Tree recursion
    - multiple recursive calls in a single case
- Recursive vs. Iterative Processes
  - All of them are implemented with recursion!
  - Recursive processes aren't done at the base case
  - Iterative processes keep their answer with them

Base Cases

How many ways can you count

- $0.00 in change using 5 types of coins?
  - 1 way
- -$0.20 in change using 5 types of coins?
  - 0 ways
- $1.00 in change using 0 types of coins?
  - 0 ways

count-change recursive cases

Blank 1 Blank 2

```
(define (cc amount kinds-of-coins)
…
(+
   (cc
        (- amount (first-denomination kinds-of-coins))
        kinds-of-coins)
   (cc
        amount
        (- kinds-of-coins 1)))))
```

```
(define (count-change amount)
  (cc amount 5))
(define (cc amount kinds-of-coins)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (= kinds-of-coins 0)) 0)
        (else (+
               (cc (- amount
                      (first-denomination kinds-of-coins))
                   kinds-of-coins)
               (cc amount
                   (- kinds-of-coins 1))))))

(define (first-denomination kinds-of-coins)
  (cond ((= kinds-of-coins 1) 1)
        ((= kinds-of-coins 2) 5)
        ((= kinds-of-coins 3) 10)
        ((= kinds-of-coins 4) 25)
        ((= kinds-of-coins 5) 50)))
```

Woah! This is overwhelming!

evens2 Solution

```
(define (evens2 sent)
  (define (evens-iter sent answer)
    (if (empty? sent)
        answer
        (evens-iter
            (bf sent)
            (se answer (first sent)))))
  (evens sent '()))
```

factorial version 2 Solution

```
(define (factorial n)
  (define (fact-iter n answer)
    (if (= n 1)
        answer
        (fact-iter (- n 1) (* n answer))))
  (fact-iter n 1))
```

Recursion (Cont.) Solution

R5C3 = R4C2 + R4C3

(R,C) = (R-1,C-1) + (R-1, C)

```
(define (pascal row col)
  (cond
    ((= col 0) 1)
    ((= col row) 1)
    (else (+ (pascal (- row 1) (- col 1))
             (pascal (- row 1) col)))))
```

Does this generate:

A) A recursive process

B) An iterative process