CS1 Lecture 15 Feb. 20, 2017

- HW4 due Wed. 2/22, 9 am
  - comment on Q1 list "format"
  - Q2: find *any* solution. Don't try to find the best/optimal solution or all of them
  - Q3: although you cannot use a loop in Q2, you may use a loop in Q3 (but function must also be recursive)
- Exam 1, Thursday evening, 2/23, 6:30-8:00pm, MacBride Auditorium
  - You must bring ID
  - For people with conflicts, email will be sent right after class
- No points in tomorrow's discussion sections. Not required. Consider them just extra office hours to help with HW4. If you are in a Wed. section, you can go to any Tues. section for help (or regular office hours)
- Wednesday: exam review

Last time

- comments/advice on use of functions
  - functions that return values, functions that have side effects or print
- Started recursion (end of Ch 5)

Today

- A few words on Exam 1
- More recursion examples

Recursion (end of Ch 5)

- Very important and useful concept
- Not just for programming, but math and everyday life, nature, etc.
- Has undeserved reputation among some people: “recursion is bad – recursive programs are inefficient” Yes, one can write very bad recursive programs but this is true of non-recursive programs as well. And recursion *can* be super useful.

Recursion

- Recursive function: function that contains within its definition calls to itself

- Consider math's factorial. E.g. 3! = 3 * 2 * 1
- You might be used to definition like: n! = n * (n-1) * … * 2 * 1
- But more precise mathematical definition of factorial function is:

  factorial(1) = 1
  factorial(n) = n * factorial (n-1), for all n > 1

- Programming-wise, can very directly translate recursive mathematical definitions into code:

```
def factorial(n):
    if (n == 1):
        return 1
    else:
        return n * factorial (n – 1)
```

- DON'T let the function call, factorial(n-1), scare you. It's just a function call. If you draw stack frames like we did in earlier lectures, it all works out fine.
- DO need to think carefully when writing/analyzing recursive programs though …

Important rules for recursive functions

- When writing a recursive function:
  - MUST have base case(s), situations when code *does not* make recursive call.
  - MUST ensure that recursive calls *make progress toward base cases.* I.e. you need to convince yourself that recursive call is “closer to” base case than the original problem you are working on
  - SHOULD ensure you don’t unnecessarily repeat work. Ignoring this contributes to recursion’s bad reputation. E.g. direct recursive implementation of Fibonacci is extremely and unnecessarily inefficient

Ch3: Stack frames

```
def countDown(n):
    if n == 0:
        print(“Blast off!”)
    else:
        print(n)
        countDown(n-1)
>>> n = 100
>>> y = 2
>>> countDown(y)
 2
 1
 Blast Off!
```

countDown: n 0

countDown: n 1

countDown: n 2

main: n 100
y 2

Recursion examples

- printList, printListReverse
- sumListItems
- isPalindrome
  - "wasitacaroracatisaw" is a palindrome
  - "sitonapotatopanotis" is a palindrome
- Fibonacci sequence: 1, 1, 2, 3, 5, 8, …

  fib(0) = 1, fib(1) = 1

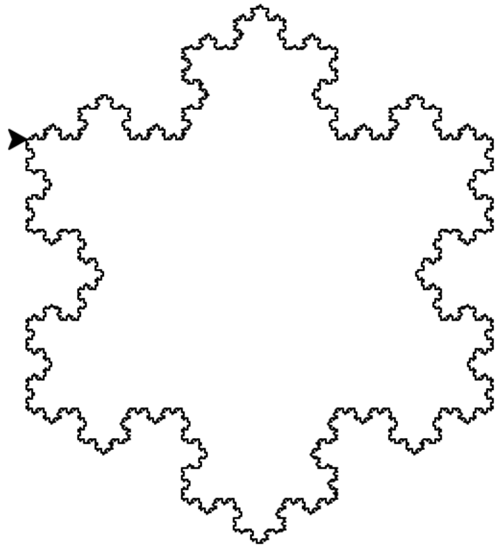
  fib(n) = fib(n-1) + fib(n-2), for n > 1
- "flatten" a list
  - E.g. [[[[[3,[2,4]]]], 0], ['a']], 23] -> [3,2,4,0,'a', 23]
- generate a number sequence
- Towers of Hanoi problem

lec14.py, lec15.py
ToHcomplete.py

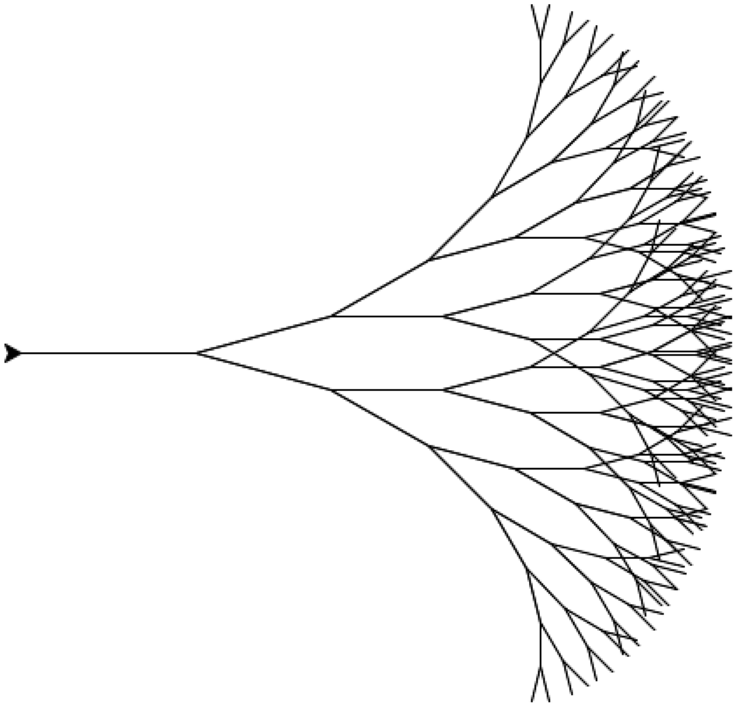
Towers of Hanoi solution

- Rods/towers A, B, C. Goal is to move disks from A to C, one at a time, never allowing larger disk to be on top of smaller disk
- Algorithm:
  - Move n-1 disks from A -> B (by this algorithm!)
  - Move final disk from A -> C
  - Move n-1 disks from B -> C (by this algorithm!)

Recursion Examples



koch.py



pic.py

Next Time

- Review for exam, and sample problems