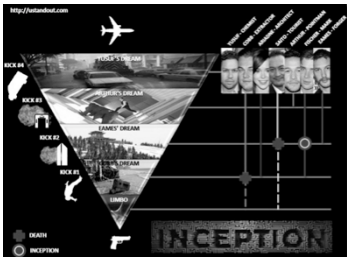
UC Berkeley EECS
Sr Lecturer SOE
Dan Garcia

The Beauty & Joy of Computing

Lecture #8
Recursion

GO SEE INCEPTION!

The coolest movie last year highlights recursion, and it was up for best picture. If you haven't seen it yet, you should, because it will help you understand recursion!!



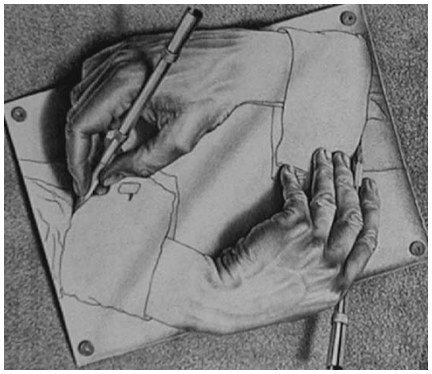
New Rule: Use scratch paper in lab!
The problems there are hard enough that you won't be able to keep it in your head!

www.worldofescher.com/gallery/A13.html

Overview

- **Recursion**
  - Demo
    - Vee example & analysis
    - Downup
  - You already know it
  - Definition
  - Trust the Recursion!
  - Conclusion

M. C. Escher : *Drawing Hands*



"I understood Vee & Downup"

a) Strongly disagree
b) Disagree
c) Neutral
d) Agree
e) Strongly agree

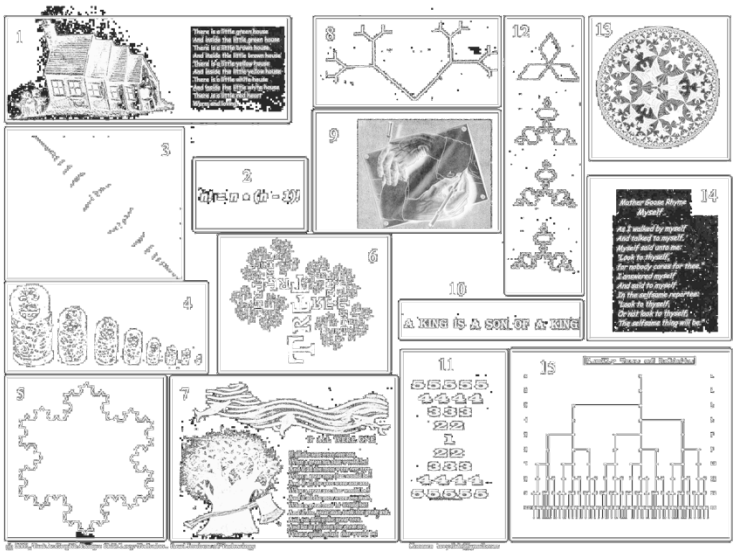
M. C. Escher : *Fish and Scales*

Definition

www.catb.org/~esr/jargon/html/R/recursion.html
www.nist.gov/dads/HTML/recursion.html

- **Recursion: (noun) See recursion.** ☺
- ***An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task***
- **Recursive solutions involve two major parts:**
  - Base case(s), the problem is simple enough to be solved directly
  - Recursive case(s). A recursive case has three components:
    - Divide the problem into one or more simpler or smaller parts
    - Invoke the function (recursively) on each part, and
    - Combine the solutions of the parts into a solution for the problem.
- **Depending on the problem, any of these may be trivial or complex.**

You already know it!



Trust the Recursion

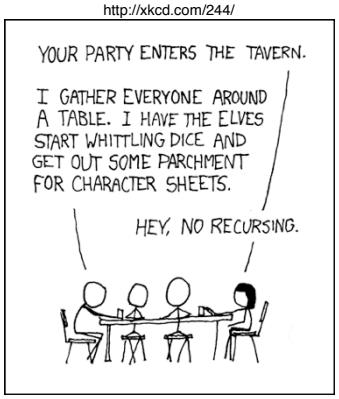
- **When authoring recursive code:**
  - The base is usually easy: "when to stop?"
  - In the recursive step
    - How can we break the problem down into two:
      - A piece I can handle right now
      - The answer from a smaller piece of the problem
    - Assume your self-call does the right thing on a smaller piece of the problem
    - How to combine parts to get the overall answer?
- **Practice will make it easier to see idea**

Sanity Check…

- **Recursion is ■ Iteration (i.e., loops)**
- **Almost always, writing a recursive solution is ◆ than an iterative one**

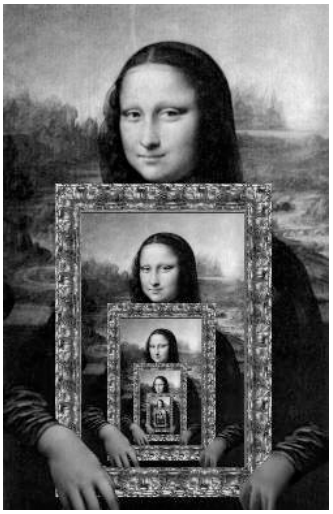
a) more powerful than, easier
b) just as powerful as, easier
c) more powerful than, harder
d) just as powerful as, harder

http://xkcd.com/244/



Summary

- **Behind Abstraction, Recursion is probably the 2nd biggest idea about programming in this course**
- **It's tremendously useful when the problem is self-similar**
- **It's no more powerful than iteration, but often leads to more concise & better code**



http://www.dominiek.eu/blog/?m=200711