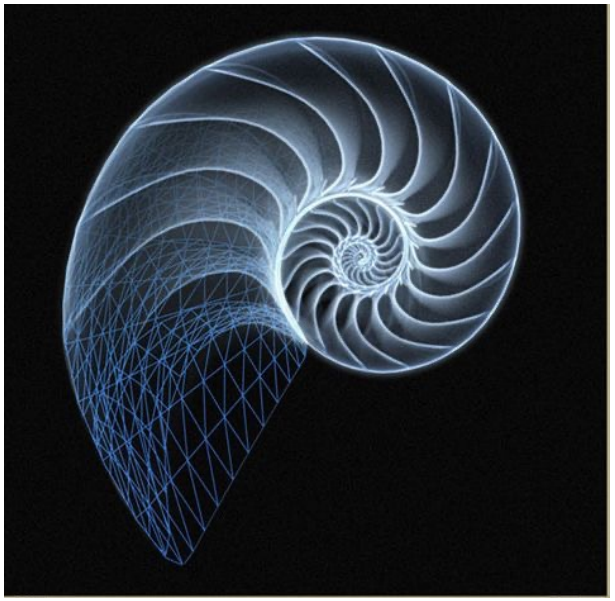
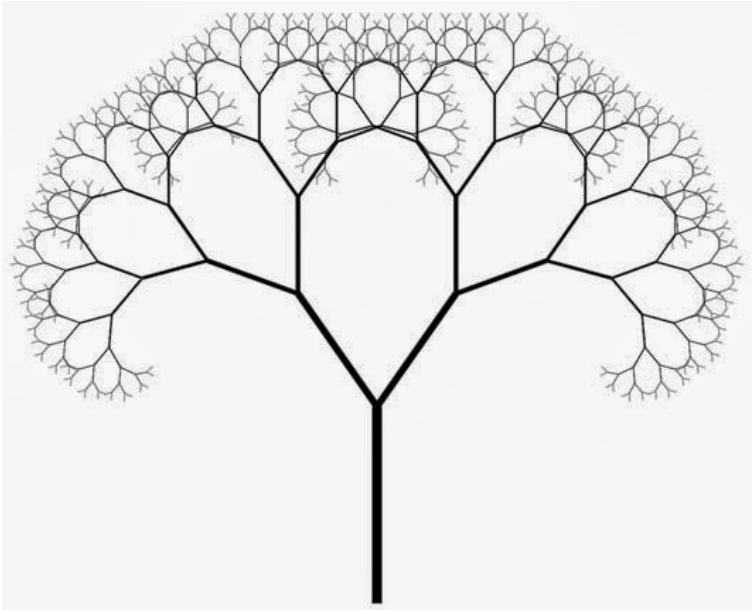
Lecture 9: Recursion

CS 51P

October 4, 2023

Recursion





What is recursion?

- Wikipedia: “Recursion occurs when a thing is defined in terms of itself.”

- Object containing smaller copies of itself

Recursion in programming

- A function calls itself by passing a different arguments
- A powerful substitute for loops
- A technique for tackling a complicated problem by taking one bite of the problem at a time
  - Divide and conquer
- Short code and easy to understand

How many students in a row?

- Loops or iteration
- Walk around and count
- The first student looks back, counts and reports

How many students in a row?

- Don’t walk around
- Don’t look back
- Don’t rely on one person (minimize each student’s amount of work)
- Don’t use indexing

How many students in a row?

- Recursion

- Ask behind “how many people are sitting behind you?”
  - If no response, then answer 0
  - If the student behind you says “let me check”, you will wait for their response. The student behind you will recur to ask behind “how many people are sitting behind you?”
  - When a response (e.g., N) is received, respond (N + 1) to the person who asks this question (e.g., the one in front of you)

Two main components of recursion

1. Base case:
   - The simplest version of your problem that all other cases reduce to
   - An occurrence that can be answered directly

   - What is the base case of the demo?

2. Recursive case:
   - The step where you break down more complex versions of the task into smaller occurrences
   - Cannot be answered directly

   - What is the recursive case of the demo?

Recursion summary

- Reduce problem into repeated, smaller tasks of the same form
- Recursion has 2 main parts: base case and recursive case
- Solution is built up as you come back up the call stack

Example

```
def mystery(n):
1   if n == 1:
2     return 1
3   else:
4     return n + mystery(n-1)
```

- what is returned by `mystery(3)`? `mystery(5)`? `mystery(k)`?

Exercise

- The number n factorial, n! in math notation, is defined as:
  - n! = n * (n-1) * (n-2) * … * 2 * 1

- For example:
  - 3! = 3 * 2 * 1 = 6
  - 5! = 5 * 4 * 3 * 2 * 1 = 120
  - 0! = 1 (by definition)

- Define a function called `factorial`, which takes in an integer number `n` as input, and returns n!

Exercise

- Fibonacci numbers:

  1, 1, 2, 3, 5, 8, 13, 21, ...

- Define a function `fib` which takes a parameter `n` (an int) and returns the n^{th} Fibonacci number