Recursion

April 13, 2020

Administrative Notes

We'll go over Exam 2.

Project 2 is out. Design is due April 20; project is due April 27. We'll spend some time talking about it.

Lab #8 will be released tonight. It will be due Thursday, April 16 before midnight.

Homework #6 is on recursion. It will be released next week. It will be released on the 20th and due on the 27th. So get a good start on Project 2, so that you don't have both of those assignments hanging over you. This will be the LAST homework.

The rest of the semester - updated schedule

This week - April 13 & 15 - Recursion, plus Lab 8 and the start of Project #2.

Next week - April 20 & 22 - A departure from the previous schedule. Jupyter notebooks and graphics with ggplot. Useful tools for the rest of the work you'll do in college. Also, Lab 9, Homework 6, and the end of Project #2.

April 27 & 29 - Binary, Decimal and Hexadecimal numbering systems. Algorithms for sorting and searching. Project #3 (using Jupyter notebooks to solve problems & report on your solution) released. Lab 10

May 4 & 6 - Analysis of algorithms. Algorithmic complexity, Big O notation. Lab 11.

May 11 - last class; review for final. Project #3 due.

Python functions

We mentioned earlier that a function can be anywhere in a Python program except on the left-hand side of an assignment statement.

Functions can come before the code that calls them. Functions can come after the code that calls them. Functions can be inside of other functions (don’t do that in 201, though!)

Functions can even call themselves.

But why would you do that? And wouldn’t that cause an infinite loop, where a function would never actually end because it just keeps calling itself?

Recursion

When a function calls itself, that's called "recursion" or "recursive programming." And it turns out to be a very useful way to solve certain problems. Like, those where certain things have to be done over and over, with only slight differences.

Up until now, when we've had things that needed to be done repeatedly, we used loops. Either for loops, or while loops.

> Using loops is called "iteration" or "iterative programming."

Every programming problem that can be solved with recursion can also be solved with iteration!!

So why use recursion? Sometimes it just makes the problem easier to solve

So how does recursion work?

Somebody gives you a task to do.

If the task is easy enough, you just do the task yourself. *This is called the "base case"*

If the task is too hard to do directly, you offer to do a part of the task, then get somebody else to do the rest of it for you. *This is called the recursive case.*

This is only gonna work if you actually do part of the task yourself, and get other people to do a simpler version of the task. Otherwise we're in an infinite loop.

An example: bring me my new laptop!!!

My new laptop is in, but not here yet. Can you bring it to me? It’s ten miles away. You have to go on foot and carry it.

Iterative solution:

- Go one mile. Then go another mile. Then go a third mile. Keep doing this until you’ve run all ten miles.
- Pick up my laptop. Come back one mile. Then come back another mile. Then come back a third mile. Keep doing this until you’ve brought my laptop back.

It works - I get my laptop! But it’s a lot of work.

The recursive version

You offer to go one mile. Then you ask somebody else to go get the laptop, which is 9 miles away.

She goes one mile, then asks somebody else to go get a laptop, 8 miles away.

This repeats until somebody is asked to go get a laptop that's only one mile away. That seems reasonable, so the last person just goes and gets the laptop and brings it one mile back.

Then each other person brings the computer back the one mile they came.

I got my computer. Nobody had to run 20 miles round trip; nobody covered more than two miles!

Notes on this example:

The *base case* was the last person, who agreed that one mile was really not too far to go to get the computer, and she just did it.

Everybody else was part of the *recursive case*. Make the problem a little bit simpler, and then get somebody else to do it. Each person had to make the problem a little simpler; that is, get closer to the laptop. If I asked you to go 10 miles, and you asked your friend to go 10 miles, and she asked her friend to go 10 miles… **we're never going to get that computer!!!**

This seems fake

It depends on having friends who happen to be one mile closer to the computer and who are willing to help out. What are the odds of that?

Okay, it’s kind of a hokey example, but:

- What if those ‘friends’ are clones - copies of you?
- And what if by ‘you’ we mean a software function that can clone copies of itself at will?

Now the example’s not all that hokey, is it?

A software example: computing a factorial

Mathematically, n factorial (written as n!) for any positive integer n is just the product n * (n-1) * (n-2)*...* 1.

We know how to solve it iteratively. Now let's do it recursively

Iterative

```
def fact(n):
    prod = 1
    for i in range(n,1, -1):
        prod *= i
    return prod
```

Recursive

```
def fact(n):
    If n == 1:
        return 1
    else:
        return (n * fact(n-1))
```

Visualizing recursion

Let’s look visually at what’s happening when the recursive function is executing:

http://www.pythontutor.com/visualize.html#mode=edit

Another example - the Fibonacci sequence

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

After the first two numbers, each number is the sum of the previous two numbers

That is, f(n) = f(n-1) + f(n-2)

Iterative

```
def fib(n):
   If n<= 3:
        return n
   Else:
        fib = [1,1]
        for i in range(3,n+1)
            fib.append(fib[i-1] + fib[i-2])
        return (fib[n])
```

Recursive

```
def fib(n):
   if n < 3:
      return 1
   else:
      return (fib(n-1) + fib(n-2))
```

Tracing the recursive routine

http://www.pythontutor.com/visualize.html#mode=display