Recursion

November 12

Administrative notes

We’ll go over test 2 tonight

We’ll talk about homework #6 at the end of lecture

Note the opportunity for extra credit - all you have to do is actually do the last 3 labs

Recursion

Recursion is solving a problem by having a function call itself, passing a smaller or simpler part of the problem.

Factorial: n! = n * n-1 * n-2 *...* 1

6! = 6 * 5 * 4 * 3 * 2 * 1 = 720

n-1! is a simpler problem than n! - it's one less number to multiply. So we can solve the problem recursively:

$$n! = n * (n-1)!$$

Requirements for recursion

You need three things for a recursive function:

- A ***base case*** - the special case where you terminate the recursive calls because this is a simple enough answer.
  - 1! = 1 and there's no need to go any further
- A ***recursive case*** - where you make the problem simpler and call the function again - n! = n * (n-1)! If n > 1
- The ***recursive call*** - where you call the function with the new arguments, and then use the returned value

Now, the recursive implementation of factorial

Recursion vs. Iteration

We can implement n! using a loop - “iteration”

(coding example of iterative_factorial)

So why would we ever use recursion? Why not just use iteration for everything?

You *could*. But some problems are easier to think about recursively, because you can break them down into pieces that are easier to think about.

Example: You are assigned to write a 100 page paper

The iterative solution:

- Write a page
- Write another page
- Keep going until your page count is at 100

This can be done, but it just seems like a lot of work

The recursive solution:

- If the assignment is only to write a one page paper, go ahead and do it.
- Ask your friend to write a 99 page paper, and write one page to add to it.
- That friend will ask another friend to write a 98 page paper, and add her page to it.

Each “friend” is only going to wind up writing one page, which might seem easier

Example: Fibonacci sequence

The Fibonacci sequence starts with 1 and 1. Then the next number is always the sum of the latest two numbers in the sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610…

Now, let’s write a function to generates the first n numbers in the Fibonacci sequence. We’ll do this both iteratively and recursively

FIbonacci Sequence: iteration and recursion

```
def iterative_fibonacci (number):
    result = [1,1]
    for i in range(2,number+1):
        next_fib = result[i - 2] + result[i-1]
        result.append(next_fib)
    return(result[number])
```

```
def recursive_fibonacci(number):
    if number <= 1:
        return(1)
    else:
        return
(recursive_fibonacci(number - 1) +
recursive_fibonacci(number - 2))
if __name__ == "__main__":
    print(iterative_fibonacci(8))
    print(recursive_fibonacci(8))
```

One more example: determine whether a string is a palindrome (is the same forwards or backwards)

```
def iterative_palindrome(saying):
    palindrome = True
    i = 0
    while (palindrome) and i <= (len(saying)/2):
        if saying[i] != saying[-(i+1)]:
            palindrome = False
        i += 1
    return(palindrome)
```

```
def recursive_palindrome (saying):
    if len(saying) < 2:
        return(True)
    if saying[0] != saying[-1]:
        return(False)
    return(recursive_palindrome(saying[1:-1]))
if __name__ == "__main__":
    saying = "amanaplanacanalpanama"
    print(iterative_palindrome(saying))
    print(recursive_palindrome(saying))
```

Things to remember:

Recursion requires: base case; recursive case; recursive call

Remember that you must always have a base case - a case that terminates recursion!

Otherwise your program will run forever!!

Your recursive call must always be on a simpler or smaller version of the problem, or again you’re going to have an infinite loop

Homework #6

Part 1 - remember that you can do division by subtraction. Just count how many times you subtract. The remainder/modulus doesn't count in this problem

20 // 6: 20 - 6 = 14; 14 - 6 = 8; 8 = 6 = 2; stop; we have done 3 subtractions so 20//6 = 3.

HW6 part 2

list.extend() - useful for HW#6

```
first_list = [1,2,3,4]
second_list = [5,6,7,8]
first_list.extend(second_list)
print(first_list)
```

What's the difference between append() and extend()? They often produce the same results, but append adds the entire argument to the initial list in one operation, while extend() iterates over the argument and adds one element at a time to the first list.

HW6 part 3

- “Hidden messages” MUST match on case!! If it’s the same word but a different case it doesn’t count!!!