Recursion, Part 2

April 15, 2020

Administrative Notes

Lab 8 - note the update to the location of the file hailstone.py

Reminder that Lab 8 is due Thursday night before midnight

Project #2 - questions?

Final Exam - Friday, May 15 - 6 - 8 pm.

The current plan is to have the test on Blackboard at that time. It will be similar to the way we did Exam 2.

I'll keep you posted if anything changes

Monday’s lecture??

Just what the heck were we “visualizing,” anyway?

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

How Python Works

Python, like many (but not all) programming languages, uses a stack of frames to keep track of where it is in the program

- What instruction gets executed next
- What variables are in scope and can be accessed
- Where to go when this function finishes

This determines exactly how the program will work, and what statement will be executed after the current statement is executed

Only the frame on TOP of the stack can be executing!!

A stack is a data structure

A way of organizing data, with a defined set of rules about what can be done.

- Similar to a list or a dictionary, but with different operations

With a stack, you can only get the most recently added item from it.

- You can think of it more literally as a stack of papers where the most recently called functions information can be found on top.
- Here’s an illustration

https://www.cs.usfca.edu/~galles/visualization/SimpleStack.html

Frames and the Python Stack

A ***frame*** is the set of all symbols (variables, constants, function names) currently in scope - currently known to the Python interpreter

When the program starts, it pushes the main program’s frame onto the stack, and the main program executes.

When the main program calls a function, Python creates a new frame for that function and pushes that frame onto the stack

- Since the function’s frame is on top of the stack, that function is now executing

When a Function Ends

When a function ends, its frame is popped off the stack

- “Pop” in this sense means remove the top element from the stack
- All its parameters and local variables disappear
- Program Control returns to the frame that’s now on top of the stack - that is, whoever called that function!!

When does a function end?

- When a return statement is executed
- When there is no return statement, but all code in the function has been executed

When does it all end?

When the main program’s code has been executed, the main program’s frame is popped off the stack and that’s the end of it. There’s no place to return control, so the program is over

- This assumes there were no errors that ended your program prematurely

Back to Recursion - 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

How do you solve a problem using recursion?

1. What is the base case?
2. How do I describe a subproblem of my problem
    a. If I repeated making that subproblem, do I get a base case?
    b. At this point we should TRUST the recursive calls
3. Assuming I have the solution to the subproblem, ***HOW DO I SOLVE MY PROBLEM WITH IT?***
    a. I should be careful to make sure I’m returning the answer to MY PROBLEM

Palindromes

Calculate whether a string is a palindrome using recursion

What’s the base case?

What about the recursive case?

Other problems that can be easily solved using recursion

Exponentiation - given x and y; calculate x^y

Base case:

Recursive case

Summing the members of a list

Base case

Recursive case