Recursion (Part 2)

April 7, 2021

Administrative Notes

Remember that Project 1 is due next Monday, April 12

Test 2 is next Wednesday, April 14

- During class time
- On Blackboard
- Looking a lot like Exam 1 - roughly 10 T/F or Multiple Choice questions, about 5 short answer questions and two programming questions
- It covers everything up through and including today's lecture. It is cumulative but focuses on the material since Exam 1
- There will be a sample exam on Blackboard; we will go over it during class on Monday

Project 1 notes

- Apparently the video I shared is causing confusion - my apologies
- You are to implement the rules as explained in the project assignment; NOT as in the video
    - I tried to point out the differences while I shared that video; I clearly didn’t do that well enough

More about Recursion

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

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

Stack - stack of plates. Put a plate on top (‘push’); take a plate off the top (‘pop’)
LIFO - last in, first out

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 < 2:
    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? #there may be more than one
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?

- A string that is zero characters long - an empty string - IS a palindrome
- A string that is one character long IS a palindrome
- A string where the first character is DIFFERENT from the last character is NOT a palindrome
    - “cat” is NOT a palindrome
-

Recursive Case

What about the recursive case?

- IF the first character is the SAME as the last character, the string IS a palindrome if what's left when you throw away the first and last characters is a palindrome

yay - remove first character; remove last character; look at what's left

- a - IS a palindrome

Pseudocode

To determine that

x= some_arbitrary_string

IS a palindrome

Check that the first character equals last character

- Create the substring new_x by throwing away the first and last character
    - new_x = [1:-1] or (x[1:len(x)-1])
- Your answer for x is the same as the answer for this substring new_x

Example: is Y = ‘tt’ a palindrome?Throw away first and last character - we have nothing left - we have an empty string left

Example: is x = ‘yay’ a palindrome?Throw away first and last character - only the ‘a’ is left and that’s a palindrome

Example: is x = ‘cat’ a palindrome
First letter does not equal last letter; NOT a palindrome

Now, let’s write this code