CS1210 Lecture 20 Oct. 8, 2021

- Quiz 2 should be graded by Monday. On Monday, I'll talk about grades so far, options for people who didn't do well on one of them or who missed Q2.
- HW 5 will be published at the end of class. Due next Friday
- Next quiz is Oct. 29, and will cover recursion (our topic today and the next few days) and objects/classes/ inheritance

Today

- Fun application of image manipulation tools discussed in last Wednesday's lecture - simple steganography
- Start important topic of Recursion (Ch 16)

Additional exercises / resources

- codingbat.com – small should-be-easy auto-graded exercises. *DO THEM ALL*
- pythontutor.com – visualize/trace code execution
- Others? hackerrank.com?

Programming games that I think can be helpful

- Human Resource Machine
  - Sequel: 7 Billion Humans
- Cargobot - original a phone app but now seems fully playable for free at: http://i4ds.github.io/CargoBot/?state=1
- Shenzhen I/O (but recommend starting with the ones above)

Recursion – Ch 16

- Very important and useful concept
- Not just for programming, but math and even everyday life, legal definitions, etc.
- Has undeserved reputation among some people: “recursion is bad – recursive programs are inefficient” Yes, one can write very bad recursive programs but this is true of non-recursive programs as well. And recursion *can* be super useful.

Recursion

- Recursive function: function whose definition contains references to/calls to itself

- For example, math's factorial (3! = 3 * 2 * 1)
  - You might be familiar with informal definition like: n! = n * (n-1) * … * 2 * 1
  - But more precise mathematical definition is:

    factorial(1) = 1

    factorial(n) = n * factorial (n-1), for all n > 1

- Programming-wise, can very directly translate recursive mathematical definitions into code:

```
def factorial(n):
    if (n == 1):
        return 1
    else:
        return n * factorial (n – 1)
```

- DON'T let the function call, factorial(n-1), scare you. It's just a function call. If you draw stack frames like we did in earlier lectures, it all works out fine.
- DO need to think carefully when writing/analyzing recursive programs though …

Important rules for recursive functions

- When writing a recursive function:
  - MUST have *base case*(s), situations when code *does not* make recursive call.
  - MUST ensure that recursive calls *make progress toward base cases.* I.e. you need to convince yourself that recursive call is "closer to" base case than the original problem you are working on
  - SHOULD ensure you *don't unnecessarily repeat work*. Ignoring this contributes to recursion's bad reputation. E.g. direct recursive implementation of Fibonacci is extremely and unnecessarily inefficient

Ch16: Recursion and stack frames

```
def factorial(n):
    if n == 1:
        result = 1
    else:
        recResult = factorial(n-1)
        result = n * recResult
    return result

>>> n = 100
>>> y = 3
>>>  answer = factorial(y)
```

factorial: n 1
recResult
result 1

factorial n 2
recResult 1
result 2

factorial n 3
recResult 2
result 6

main: n 100
y 3
answer 6

Recursion

More basic recursion examples (lec20.py)

- Print the items of a list, one per line
- Print the items of a list, one per line, in reverse order
  - Idea?
  - Consider list as: theFirstItem <the rest of the list>
  - Reverse is: reverse(<the rest of the list>) theFirstIte
  - Consider list as: <list from start to near end> theLastItem
  - Reverse is: theLastItem reverse(<list from start to near end>)
- return a string that is the reverse of the given string
- sum the items in a list
- return True/False depending on whether given string is a palindrome (e.g. Was it a car or a cat I saw?)
- return num of digits in an integer
- return sum of digits in an integer
- return string with each occurrence of a particular character replaced with a new character
- return count of number of substrings that have same first and last characters
- compute nth Fibonacci number:
  - 1, 1, 2, 3, 5, 8, 13, …
  - Definition: fib(1) = 1, fib(2) = 1,

    fib(n) = fib(n-1) + fib(n-2) for n > 2

Ch16: Recursion and stack frames

```
# Fibonacci numbers: 1,1, 2, 3, 5, 8, 13, …
# variables fnm1, fnm2, result are used just so that it's a little easier to follow
# what's going on when stepping through execution/viewing stack frames
#
def fib(n):
        if (n == 0) or (n == 1):
            result = 1
        else:
            fnm1 = fib(n-1)
            fnm2 = fib(n-2)
            result = fnm1 + fnm2
        return result

>>> fib(4)
3
```

Copy/paste code above and execute step-by-step on Pythontutor.com to watch variables and stack diagrams

HW5

- First two are “basic” recursion problems.
- Q3 is hard. Okay to use a loop in the function as long as you also use recursion

Next time

- More on recursion