CS1 Lecture 14 Feb. 17, 2017

- HW4 due Wed. 2/22, 9 am
- Exam 1, Thursday evening, 2/23, 6:30-8:00pm
- Solutions for HW2, HW3 and DS4 have been posted.
- A few words about course grades
- Will review for exam next Wednesday
  - types of problems, samples problems, etc.

About course grades

People ask if course is "curved"

answer: no and yes

1. The "no" part: if everyone does excellent work and demonstrates mastery of all the material, then I am happy to give everyone an A.
2. There are no pre-set point/percentage -> course grade mappings like many of your are familiar with from high school (e.g. A for 90-100, B for 80-90, etc.). Instead, based on my sense of how well students mastered the material, I map score ranges to grades. Exam averages are often relatively low in a class like this - 60-75%. People who get A's in the class generally do well on all the homework 80-100%, and 80% or more on exams.
3. The "yes" part: the College of Liberal Arts has some guidelines about grade distribution (15% A, 34% B, 40%D, etc.). This is not a rule. Item 1 above takes precedence! I can give all A's (or F's). You are *not* competing with other students. Your job is to master the material. However, it *is* often the case that the grade distribution "curve" fits the CLAS percentages approximately well. That's simply because 10-20% of the people do very well, 30-40% do well, etc. Many people who get Cs/Ds skip homeworks, don't put a lot of effort into them, etc.

After the first exam, I'll talk more specifically about numbers and where people stand so far

Last time

Finished Ch 10

- more on mutability, including lists as function arguments

Today

- Problem from last time: letterCounts
- final pre-exam topic – recursion (end of Ch 5)

Functions with/without side effects

- Some functions compute something and return a value without *side effects*. That is, they do any output and don't change the values of any objects that exist outside of the execution of that function.
- Other functions do have side effects. They either print something (or affect GUI elements) or *change values of objects that exist outside the function execution*. Such functions often don't return anything. And such functions can maybe helpfully be thought of as commands.

Thoughts on this function?

```
# Given a list of 1 or more numbers or strings, return the smallest
# and largest items.
#
def getSmallestAndBiggest(listOfStuff):
    listOfStuff.sort()
    return (listOfStuff[0], listOfStuff[-1])
```

Functions should not modify their input unless the function specification explicitly says to do so!

```
# Given a list of 1 or more numbers or
# strings, return the smallest and largest
# items.
#
def getSmallestAndBiggest(listOfStuff):
    sortedCopyOfList = sorted(listOfStuff)
    return (sortedCopyOfList[0], sortedCopyOfList[-1])
```

```
# Given a list of 1 or more numbers or
# strings, sort the list and return the smallest and
# largest items.
#
def sortAndGetSmallestAndBiggest(listOfStuff):
    listOfStuff.sort()
    return (listOfStuff[0], listOfStuff[-1])
```

Practice problem mentioned last time

Lists make it easy to generalize the printVowelInformation(inputString) function of HW2.

How would you implement printLetterCounts(inputString, letters) that prints the number of occurrences in inputString of each letter in letters?

```
>>> printLetterCounts("This is a sentence containing a variety of letters",
                      "aeiouy")
'This is a sentence containing a variety of letters' has:
    4 'a's
    6 'e's
    5 'i's
    2 'o's
    0 'u's
    1 'y's
    and 32 other letters
```

lec13letterCountsStart.py, lec14letterCounts.py

Recursion (end of Ch 5)

- 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 used to def 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

Ch5: Recursion and stack frames

```
def countDown(n):
    if n == 0:
        print("Blast off!")
    else:
        print(n)
        countDown(n-1)
>>> n = 100
>>> y = 2
>>> countDown(y)
 2
 1
 Blast Off!
```

countDown: n 0

countDown: n 1

countDown: n 2

main: n 100

y 2

Recursion

- More simple recursion examples
  - Print the items of a list, one per line
  - Print the items of a list, one per line, in reverse order
  - return a string that is the reverse of the given string
  - return True/False depending on whether given string is a palindrome (e.g. Was it a car or a cat I saw?)
  - 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

HW4 Q2 and Q3

- Both are recursion problems. Not super simple. Practice simpler problems first!
- Hint for Q2: suppose set of coins is 5, 8, 11. If change for amount is possible, it's one of
  - one 5 coin plus change for (amount – 5)
  - one 8 coin plus change for (amount – 8)
  - one 11 coin plus change for (amount – 11)
  - These translate directly to recursive code.
  - What about base cases? When can you directly produce answer without recursion?

Next Time: more recursion