CS1110
Lecture 11: Intro to Recursion

Prelim preparation/upcoming schedule
This week (Feb 25 – Mar 1)
- Prelim conflicts (makeup requests) at midnight on CMS
- A3 out today; short, designed to help prepare you for the exam

Labs today and tomorrow:
- Lab 5 (lists) due
- pick up your graded A2s – feedback will help you for prelim
- Lab 6 (recursion) out – not optional, but that material is not on the prelim. Due at beginning of lab session after the prelim.

Thursday: lecture (recursion II) as usual

Next week (Mar 4 – Mar 8)
- Monday Mar 4: A3 due
- Tuesday Mar 5: review session instead of regular lecture
- Labs Mar 5/6:
  - pick up your graded A2s (if you haven't already)
  - No new lab activity, optional attendance: treat as office hours, or opportunity to work more on Lab 6 (due in lab the week after)
- Thursday Mar 7:
  - “lecture” = office hours with profs (location TBA)
  - Prelim: 7:30-9pm, 116 Kennedy Hall/Call Auditorium

A2 solutions

New In-Lab Collaboration Policy
To get your questions answered in lab faster:
We (now) encourage you to talk to your table-mate or other students in lab to solve the problems you are given. You may look at each other's lab code while in lab.

"Submission petitions": new policy
Need an extension/missed a submission deadline? Please email head TA Qin Jia (qj34@cornell.edu), not the instructor(s).

*Everyone who submitted received the same "participation" grade, which enabled us to finish the grading faster.
**Except we don't care whether you wrote "end" to indicate the end of execution or not.
Nested Lists (appear in A3)

- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

```
>>> a = [7, 'k']
>>> b = [3, 6]
>>> c = ['A', 'B', b]
>>> x = [7, a, c, 5]
```


A Recursive Function

```
def num_es(s):
    """Returns: number of 'e's in s. Precond: s a string""
    if s == '':
        # case: s is empty string
        return 0
    # case: <s> has at least one char
    return (1 if s[0] == 'e' else 0) + num_es(s[1:])
```

Indeed, if s has at least one character, the number of 'e's in s is the number of 'e's in s[0] + the number of 'e's in s[1:].

How to Think About Recursive Functions

1. Have a precise function specification.
2. Base case(s):
   - When the argument values are as "small" as possible
   - When the answer is determined with little calculation.
3. Recursive case(s):
   - Verify recursive cases with the specification
4. Termination:
   - Arguments of calls must somehow get "smaller", so each recursive call gets closer to a base case

Understanding the String Example

1. Step 1: Have a precise specification
```
def num_es(s):
    """Returns: number of 'e's in s. Precond: s a string""
    if s == '':
        return 0
    # case: s is not empty
    if s[0] in string.whitespace:
        return num_es(s[1:])
    # case: s not empty and s[0] not blank
    return (1 if s[0] == 'e' else 0) + num_es(s[1:])
```

   “Write” your return statement using the specification

   Base case
   Recursive case

2. Step 2: Check the base case
   - When s is the empty string, 0 is returned. Good.

Example: Remove Blanks from a String

```
def deblank(s):
    """Returns: s with blanks removed""
    if s == '':
        return s
    # case: s is not empty
    if s[0] in string.whitespace:
        return deblank(s[1:])
    # case: s not empty and s[0] not blank
    return (s[0] + deblank(s[1:])
```

   Check the four points:
   1. Precise specification?
   2. Base case: correct?
   3. Recursive case: progress toward termination?
   4. Recursive case: correct?

   Expression: x in the list returns True if x is a member of list (and False if it is not)