Outline

- Test #1 Follow-up
- Short survey on Introductory CS Courses
  - Will receive link for the survey this Friday
  - Have until Friday, March 11
  - Check your email
- Recursion
Test #1

Mean of evening classes were a tad lower than the day class
Evening class mean is adjusted 0.5/30 higher
Overall average 67.9%
There was a lot of writing...
Post-Test Exercise

- 1% post-test exercise, follow instructions on sticker, either on last page of test paper, or an inner page if there is no room
- Exercise is due **March 2nd, 11:59pm**, NOT the date on the sticker
- In testing there was an occasional error in submitting, which is fixed by reloading the web page and continuing
Fire Alarm Incident

- About 180 students had their test interrupted by a fire alarm
- The marking scheme has no provision for make-up tests; all likely dates overlap things such as assignment due dates or other course events
- For individuals who miss a test for valid reason, we re-evaluate the mark based on the second test and final
- Consulting with the department’s undergraduate chair, we use a formula we believe neither gives an advantage nor a disadvantage to the affected students
Replace Test #1 Grade

a1: class average on test #1  
a2: class average on test #2  
ae: class average on final exam  
g2: student’s grade on test #2  
ge: student’s grade on final exam

Test #1 score:  \[ g_1 = \left( \frac{g_2/a_2 + ge/ae}{2} \right) \times a_1 \]

Rationale: Student standing the same compared to the average on test #1 as compared to the average on test #2 and the final
What about those who didn’t have a fire alarm?

Although we think the formula for those who missed test #1 gives them neither an advantage nor a disadvantage, we will offer the remaining students the maximum of either their current grade on test #1 or the grade calculated using the formula on the previous slide.

If a majority of students vote for this change, students who perform better relative to their peers on test #2 and the final may improve their test #1 grade.

*The Vote will be on March 2nd*
Summing Lists

\[ L_1 = [1, 9, 8, 15] \]
\[ \text{sum}(L_1) = ??? \]

\[ L_2 = [[[1, 5], [9, 8], [1, 2, 3, 4]] \]
\[ \text{sum}([\text{sum}(\text{row}) \text{ for } \text{row} \text{ in } L_2]) = ?? \]

\[ L_3 = [[[1, 5], 9, [8, [1, 2], 3, 4]] \]
How can we sum \( L_3 \)?
Re-use built-in ... Recursion!

- A function `sum_list` that adds all the numbers in a nested list shouldn’t ignore built-in `sum`.
- ... except `sum` wouldn’t work properly on the nested lists, so make a list-comprehension of their `sum_lists`.
- But wait, some of the list elements are numbers, not lists.
- Write a definition of `sum_list` — don’t look at the next slide yet!
def sum_list(L):
    """ (list or int) -> int
    Return L if it's an int, or sum of the numbers in possibly nested loop
    >>> sum_list(17)
    17
    >>> sum_list(1, [2, 3, [4]], 6)
    15
    ""
    # reuse: isinstance, sum, sum_list
    if isinstance(L, list):
        return sum([sum_list(x) for x in L])
    else:
        return L
Tracing Recursion

To understand recursion, trace from simple to complex:

- Trace `sum_list(17)`
- Trace `sum_list([1, 2, 3])`. Remember how the built-in `sum` works...
- Trace `sum_list([[1, [2, 3], 4, [5, 6]]])`. Immediately replace calls you’ve already traced (or traced something equivalent) by their value
- Trace `sum_list(1, [2, [3, 4], 5], 6, [7, 8])`. Immediately replace calls you’ve already traced by their value
Depth of a List

Define the depth of L as 1 plus the maximum depth of L’s elements if L is a list, otherwise 0

- The definition is almost exactly the Python code you write!
- Start by writing return and pythonese for the definition:
  ```python
  if instance(L, list):
      return 1 + max([depth(x) for x in L])
  else:  # L is not a list
      return 0
  # find the bug! (then fix it...)
- Deal with the special case of a non-list
Trace to Understand Recursion

Trace in increasing complexity; at each step fill in values for recursive calls that have (basically) already been traced

- Trace \texttt{depth([])}
- Trace \texttt{depth(17)}
- Trace \texttt{depth([3, 17, 1])}
- Trace \texttt{depth([5, [3, 17, 1], [2, 4], 6])}
- Trace \texttt{depth([14, 7, [5, [3, 17, 1], [2, 4], 6], 9])}
Maximum Number in Nested List

Use the built-in max muck like sum

- How would you find the max of non-nested list?
  \[
  \text{max}(...)
  \]
- How would you build that list using a comprehension?
  \[
  \text{max}([...])
  \]
- What should you do with list items that were themselves lists?
  \[
  \text{max}([\text{rec\_max}(x)...])
  \]
- Get some intuition by tracing through flat lists, lists nexted one deep, then two deep ...

CSC148 L5102 Spring 2016
Code for `rec_max`

```python
if instance(L, list):
    return max([rec_max(x) for x in L])
else:
    return L
```
Trace the Recursion

Trace from simple to complex; fill in already-solved recursive calls

- Trace \texttt{rec\_max}([3, 5, 1, 3, 4, 7])
- Trace \texttt{rec\_max}([4, 2, [3, 5, 1, 3, 4, 7], 8])
- Trace \texttt{rec\_max}([6, [4, 2, [3, 5, 1, 3, 4, 7], 8], 5])
Get Some Turtles to Draw

- Spawn some turtles, point them in different directions, get them to draw a little and then spawn again...
- Try out `tree_burst.py`
- Notice that `tree_burst` returns `NoneType`: we use it for its side-effect (drawing on a canvas) rather than returning some value
Base case, General case

You will have noticed that a recursive function has a conditional structure that specifies how to combine recursive subcalls (general case), and when/how to stop (the base case, or cases)

What happens if you leave out the base case?