Master recursion, this & in next week's lab.

CSC148 winter 2014

more recursion

week 4

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/F13/
416-978-5899

January 26, 2014

Computer Science
UNIVERSITY OF TORONTO

Outline

Assignment #1

Class design for cheese wrangling

Recursion on nested lists

Testing, big and small

Separation of concerns

$$\begin{tikzcd}
\text{generate sequences of moves} & \text{records + enforces rules.} & \text{displays moves + allow interaction} \\
\texttt{Tour} \arrow[u] & \boxed{\texttt{TOAHModel}} \arrow[l] \arrow[r] \arrow[u] & \texttt{ConsoleController} \arrow[u]
\end{tikzcd}$$

a relevant example

This is a job for recursion:

$$M(n) = \begin{cases} 1 & n == 1 \\ \min\left\{1 \leq i < n \mid 2 \times M(n-i) + 2^i - 1\right\} & \text{otherwise.} \end{cases}$$

tuples can be ordered, sorted, their min/max

That's a recursive formula. Python has a built-in function `min`. You probably want to combine (tuple?) the minimum number of moves with the split (i) that produces it.

nesting depth of list

[], [1, 2, 4] → 1
[1, 2, [4, 5], 6, [7]] → 2
14, "foo" → 0

Define the nesting-depth of L as 1 plus the maximum nesting 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:

```
return (1 + max([nested_depth(x) for x in L] + [0])
               if isinstance(L, list) else 0)
```

- 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 nested_depth([]) ~~(in)~~ $\rightarrow$ ~~1+max([0])~~ 1+max([] + [0]) $\rightarrow$ 1
- Trace nested_depth(17) $\longrightarrow$ 0 (not a list).
- Trace nested_depth([3, 17, 1]) $\longrightarrow$ 1 + max([0, 0, 0] + [0]) $\longrightarrow$ 1
- Trace nested_depth([5, [3, 17, 1], [2, 4], 6]) $\longrightarrow$ 1 + max([0, 1, 1, 0] + [0]) $\longrightarrow$ 1+1 $\rightarrow$ 2
- Trace nested_depth([14, 7, [5, [3, 17, 1], [2, 4], 6], 9])

maximum number in nested list

Use the built-in `max` much like `sum`

- how would you find the max of non-nested list?

  ```
  max(...)
  ```

- how would you build that list using a comprehension?

  ```
  max([...])
  ```

- what would you do with list items that were themselves lists?

  ```
  max([rec_max(x) ...])
  ```

- get some intuition by tracing through flat lists, lists nested one deep, then two deep...

trace the recursion

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

- trace `rec_max([3, 5, 1, 3, 4, 7])`
- trace `rec_max([4, 2, [3, 5, 1, 3, 4, 7], 8])`
- trace `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...

before and after coding:

Test your docstring examples automatically:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

For more thorough testing, use `unittest`