

# 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 29, 2014

# Outline

Class design for cheese wrangling

Recursion on nested lists

Testing, big and small

# Separation of concerns

Tour  $\leftarrow$  TOAHModel  $\rightarrow$  ConsoleController

## a relevant example

from AI handout

This is a job for recursion:

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

use range?

express this in python

- you probably want to bundle the min # of moves with the split (i) that achieves it.

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.

Note tuples can be compared in Python  
 $\min(\{(3,2), (1,3), (7,4), (1,2)\}) \rightarrow (1,2)$

## nesting depth of list

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([])`

$$1 + \max([0]) \rightarrow 1$$

- ▶ Trace `nested_depth(17)`  $\rightarrow 0$

- ▶ Trace `nested_depth([3, 17, 1])`

$$1 + \max([0, 0, 0, 0]) \rightarrow 1 + 0 \rightarrow 1$$

- ▶ Trace `nested_depth([5, [3, 17, 1], [2, 4], 6])`

$$1 + \max([0, 1, 1, 0]) \rightarrow 2$$

- ▶ Trace

`nested_depth([14, 7, [5, [3, 17, 1], [2, 4], 6], 9])`

$$1 + \max([0, 0, 2, 0]) \rightarrow 3$$

maximum number in nested list <sup>needed memory</sup>  $([1, [2, 4], 5, [6, 7]])$

Use the built-in max much like sum

$\max([1, 4, 5, 7])$

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

$\max(\dots)$

- ▶ how would you build that list using a comprehension?

$\max([\dots])$

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

$\max([\text{rec\_max}(x) \dots])$

*if isinstance(x, list) else x  
for x in L*

- ▶ 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])`

$$\text{max}([3, 5, 1, 3, 4, 7]) \rightarrow 7$$

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

$$\text{max}([4, 2, 7, 8]) \rightarrow 8$$

- ▶ trace

`rec_max([6, [4, 2, [3, 5, 1, 3, 4, 7], 8], 5])`

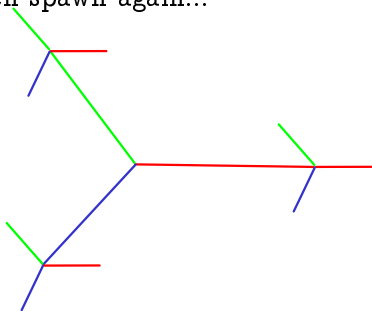
$$\text{max}([6, 8, 5]) \rightarrow 8$$



## 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`



## 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?

## before and after coding:

Test your docstring examples automatically:

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

For more thorough testing, use **unittest**