Outline

Class design for cheese wrangling

Recursion on nested lists

Testing, big and small
Separation of concerns

Tour ← TOAHModel → ConsoleController
This is a job for recursion:

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

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:

\[ \min(\{(3, 2), (1, 3), (7, 4), (1, 2)^3\}) \rightarrow (1, 2) \]
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:
  
  ```python
  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])`
  \[1 + \max([0, 0, 1, 0]) \rightarrow 1 + 0 \rightarrow 1\]

- Trace `nested_depth([5, [3, 17, 1], [2, 4], 6])`
  \[1 + \max([0, 0, 1, 0]) \rightarrow 2\]

- Trace `nested_depth([14, 7, [5, [3, 17, 1], [2, 4], 6], 9])`
  \[1 + \max([0, 0, 2, 0, 0]) \rightarrow 3\]
maximum number in nested list \( \left[ 1, [2, 4], 5, [0, 7] \right] \)

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

- how would you find the max of non-nested list?
  \[
  \text{max}(\ldots)
  \]

- how would you build that list using a comprehension?
  \[
  \text{max}([\ldots])
  \]

- what would you do with list items that were themselves lists?
  \[
  \text{max}([\text{rec\_max}(x) \ldots])
  \]
  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 \texttt{rec_max([3, 5, 1, 3, 4, 7])}
  \[
  \text{max}([3, 5, 1, 3, 4, 7]) \rightarrow 7
  \]

- trace \texttt{rec_max([4, 2, [3, 5, 1, 3, 4, 7], 8])}
  \[
  \text{max}([4, 2, 7, 8]) \rightarrow 8
  \]

- trace \texttt{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
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:

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

For more thorough testing, use `unittest`