Announcements

**Midterm:**
- Grades released on MarkUs
- Sample solutions posted on course website
- You may submit a remark request up until **Tuesday Nov 6th.** (See course syllabus). Please note that we regrade the *entire* midterm, not just one particular question.
Announcements

Assignment 1:
- Forthcoming (hopefully tonight).
Reading week is Nov 5-9!

No lecture or lab.
Office hours *by appointment* (email course address).
Last time: list comprehensions

CSC148, INTRODUCTION TO COMPUTER SCIENCE
DAVID LIU
Recall partial tracing

```python
>>> obj = [1, [2, 3], [[4], 5, 6]]
>>> nested_sum(obj)
```

<table>
<thead>
<tr>
<th>sublist</th>
<th>nested_sum(sublist)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>[2, 3]</td>
<td>5</td>
</tr>
<tr>
<td>[[4], 5, 6]</td>
<td>15</td>
</tr>
</tbody>
</table>
List comprehensions give the most important information!

\[
\text{[nested\_sum(sublist) for sublist in obj]}
\]

\[
==
\]

\[
[1, 5, 15]
\]
More complex: semi_homogeneous

A single integer and empty list are semi-homogeneous. In general, a list is semi-homogeneous if and only if:

- (all of its sub-nested-lists are integers) OR (all of them are lists)
- all of its sub-nested-lists are semi-homogeneous
Trees

CSC148, INTRODUCTION TO COMPUTER SCIENCE
DAVID LIU
Tree definition

A tree is either empty, or a root value connected to zero or more other trees, called the tree’s subtrees.
Tree attributes and recursive template

class Tree:
    _root: Optional[Any]
    _subtrees: List[Tree]

def method(self) -> ...:
    if self.is_empty():
        ...
    else:
        for subtree in self._subtrees:
            ... subtree.method() ...
Tree Deletion

MUTATING TREES!
Delete an item

def delete_item(self, item) -> bool:
    
    """Delete *one* occurrence of <item> from this tree.

    Return True if <item> was deleted, and False otherwise.
    """
A start of an implementation

def delete_item(self, item) -> bool:
    if self.is_empty():
        return False  # item isn’t in the tree
    elif self._root == item:
        self._delete_root()
        return True   # item was deleted
    else:
        ... # recurse somehow
Tree Deletion continued

MUTATING TREES!
def delete_item(self, item) -> bool:
    """Delete *one* occurrence of <item> from this tree.

    Return True if <item> was deleted, and False otherwise."""
The problem of empty trees

```python
>>> t = Tree(10, [Tree(1, []), Tree(2, []), Tree(3, [])])
>>> t.delete_item(1)
True
>>> t.delete_item(2)
True
>>> t.delete_item(3)
True
```
A hidden assumption

“self._subtrees doesn’t contain any empty trees.”
A representation invariant!

“self._subtrees doesn’t contain any empty trees.”
Trees and nested lists
Nested list representation of trees
Nested list representation of trees