TODAY

- Review: Immutable Dictionaries
- Deep Tuples
- Immutable Trees

REVIEW: IMMUTABLE DICTIONARIES

```python
>>> phone_bk = make_idict(("Ozzy", "555-5555"),
...                         ("Tony", "123-4567"),
...                         ("Geezer", "722-2284"))
>>> idict_select(phone_bk, "Ozzy")
"555-5555"
>>> idict_select(phone_bk, "Geezer")
"722-2284"
>>> idict_keys(phone_bk)
("Ozzy", "Tony", "Geezer")
```

REVIEW: HIGHER ORDER FUNCTIONS FOR SEQUENCES

```python
>>> nums = (1, 2, 3, 4, 5)
>>> tuple(map(lambda x: x * x, nums))
(1, 4, 9, 16, 25)
>>> tuple(map(lambda x: x + 1, nums))
(2, 3, 4, 5, 6)
>>> tuple(filter(lambda x: x % 2 == 0, nums))
(2, 4)
>>> tuple(filter(lambda x: x <= 3, nums))
(1, 2, 3)
>>> from functools import reduce
>>> reduce(lambda x, y: x * y, nums, 1)
120
>>> reduce(lambda x, y: x + y, nums, 0)
15
```
**Practice: Higher Order Functions for Sequences**

What are the outputs for each of the following lines of Python?

```python
>>> from operator import add
>>> tuple(map(lambda x: reduce(add, x), ((2, 3), (5, 6), (8, 9))))
(5, 11, 17)
```

```python
>>> tuple(map(lambda x: x - 1, filter(lambda x: x % 2 == 0, map(lambda x: x + 1, range(10)))))
(1, 3, 5, 7, 9)
```

**Announcements**

- Homework 5 is due **July 6**.
- Project 2 is due **July 13**.
- Project 1 contest is on!
  - **How to submit**: Submit a file pig.py with your final_strategy to proj1-contest.
  - **Deadline**: Friday, **July 6** at 11:59pm.
  - **Prize**: One of 3 copies of Feynman and 1 extra credit point.
  - **Metric**: We will simulate your strategy against everyone else’s, and tally your win rate. Draws count as losses.

**Midterm 1**

- Midterm 1 is on **July 9**.
  - **Where**: 2050 VLSB.
  - **When**: 7PM to 9PM.
  - **How much**: Material covered until July 4.
- **Closed book and closed electronic devices.**
- **One 8.5” x 11” cheat sheet** allowed.
- **Group portion** is 15 minutes long.
- **Post-midterm potluck** on Wednesday, **July 11**.

**Hierarchical Data**

Often we find that information is nicely organized into hierarchies.

Example: Writing!

```
>>> p = (["Cat", "is", "fat."],
      ["Cat", "is", "coming."],
      ["Watch", "Out!"])
```

**Hierarchical Data: Deep Tuples**

We already had a way of representing something like this.
OPERATING ON DEEP TUPLES

So we already have a simple way of organizing data into hierarchies using "deep tuples."

How do we manipulate deep tuples?
- Not that different from working with regular tuples.
- Use tree recursion!

EXAMPLE: OPERATING ON DEEP TUPLES

Let’s say I want to write a function, evens_count, that counts the number of even numbers found in a deep tuple containing numbers (or tuples).

```python
>>> woah_deep = ((1, 2), 3, ((4, 5), 6))
>>> evens_count(woah_deep)
3
```

EXAMPLE: OPERATING ON DEEP TUPLES

How would we have solved this if we were handed a simple tuple of numbers?

```python
# Iterative
def evens_count(t):
    total = 0
    for num in t:
        if num % 2 == 0:
            total += 1
    return total
```

EXAMPLE: OPERATING ON DEEP TUPLES

```python
# Recursive
def evens_count(t):
    if len(t) == 0:
        return 0
    if t[0] % 2 == 0:
        return 1 + evens_count(t[1:])
    return 0 + evens_count(t[1:])
```

EXAMPLE: OPERATING ON DEEP TUPLES

```python
def is_tuple(x):
    return type(x) is tuple
```

PRACTICE: OPERATING ON DEEP TUPLES

Write the procedure `deep_filter`, which takes a predicate and a deep tuple and returns a new deep tuple with only the items for which predicate returns True.

```python
>>> woah_deep = ((1, 2), 3, ((4, 5), 6))
>>> deep_filter(lambda x: x % 2 == 0, woah_deep)
((2,), ((4,), 6))
```

PRACTICE: OPERATING ON DEEP TUPLES

```python
def deep_filter(pred, dt):
    if len(dt) == 0:
        return dt
    if is_tuple(dt[0]):
        return (deep_filter(pred, dt[0]),) +
               deep_filter(pred, dt[1:])
    if pred(dt[0]):
        return (dt[0],) + deep_filter(pred, dt[1:])
    return deep_filter(pred, dt[1:])
```
Hierarchical Data: Trees

A tree data structure traditionally has 2 parts:
1. A datum – The data stored in the top point of the tree.
2. Some children – Any trees that appear below this tree.

Hierarchical Data: Trees

Hierarchical Data: Trees

Hierarchical Data: Trees

Hierarchical Data: Trees

I Trees

>>> fir = make_tree(1,
    (make_tree(2),
    make_tree(3,
    (make_tree(4),
    make_tree(5)))))

    1                   
    /                   
   2 - 3               
   |                  /
   4 - 5 - 6         /
   |                7 
   4 - 5 - 6         .
Example: Operating on ITrees

Suppose I want to write the function `itree_prod`, which takes an ITree of numbers and returns the product of all the numbers in the ITree.

```python
>>> t = make_itree(1, (make_itree(2),
                   make_itree(3),
                   make_itree(4)))
>>> itree_prod(t)
24
```

Idea: split the problem into 2 different parts: handling a single tree and handling a group of trees (a forest).

```python
def itree_prod(t):
    return itree_datum(t) *
    forest_prod(itree_children(t))

def forest_prod(f):
    if len(f) == 0:
        return 1
    return
    itree_prod(f[0]) *
    forest_prod(f[1:])
```

Practice: Operating on ITrees

Write the function `max_path_sum`, which takes an ITree of positive numbers and returns the largest sum you can get adding all the numbers along a path from the root to a leaf.

```python
def max_path_sum(t):
    ??????

def max_forest_sum(f):
    if len(f) == 0:
        return 0
    ??????
```
**Practice: Operating on ITrees**

Write the function `max_path_sum`, which takes an ITree of positive numbers and returns the largest sum you can get adding all the numbers along a path from the root to a leaf.

```python
def max_path_sum(t):
    max_child_sum = max_forest_sum(itree_children(t))
    return itree_datum(t) + max_child_sum

def max_forest_sum(f):
    if len(f) == 0:
        return 0
    return max(max_path_sum(f[0]), max_forest_sum(f[1:]))
```

**Conclusion**

- Organizing data into hierarchies is **very** useful and **very** common in Computer Science.
- We can think of nested tuples as a simple form of a tree structure that only has leaves.
- ITrees are useful for representing general tree-structures.
- **Preview:** binary search trees!