Discussion 03:
Sequences and Trees

TA: **Jerry Chen**
Email: **jerry.c@berkeley.edu**
TA Website: **jerryjrchen.com/cs61a**

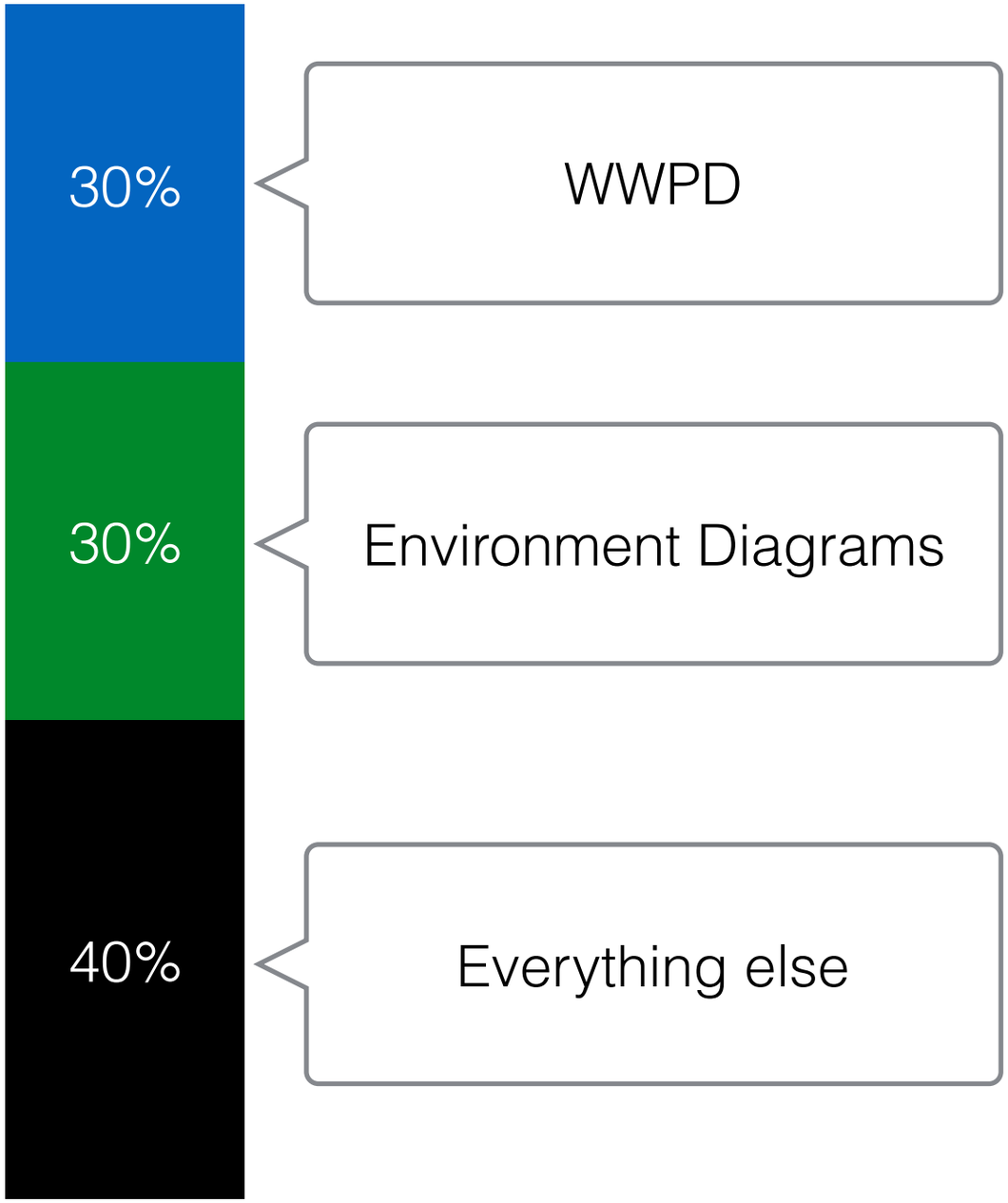
Agenda

1. Attendance
2. Announcements
3. Check Your Understanding
4. Sequences (fast)
5. Trees
6. Data Abstraction (skipped, see slides later)

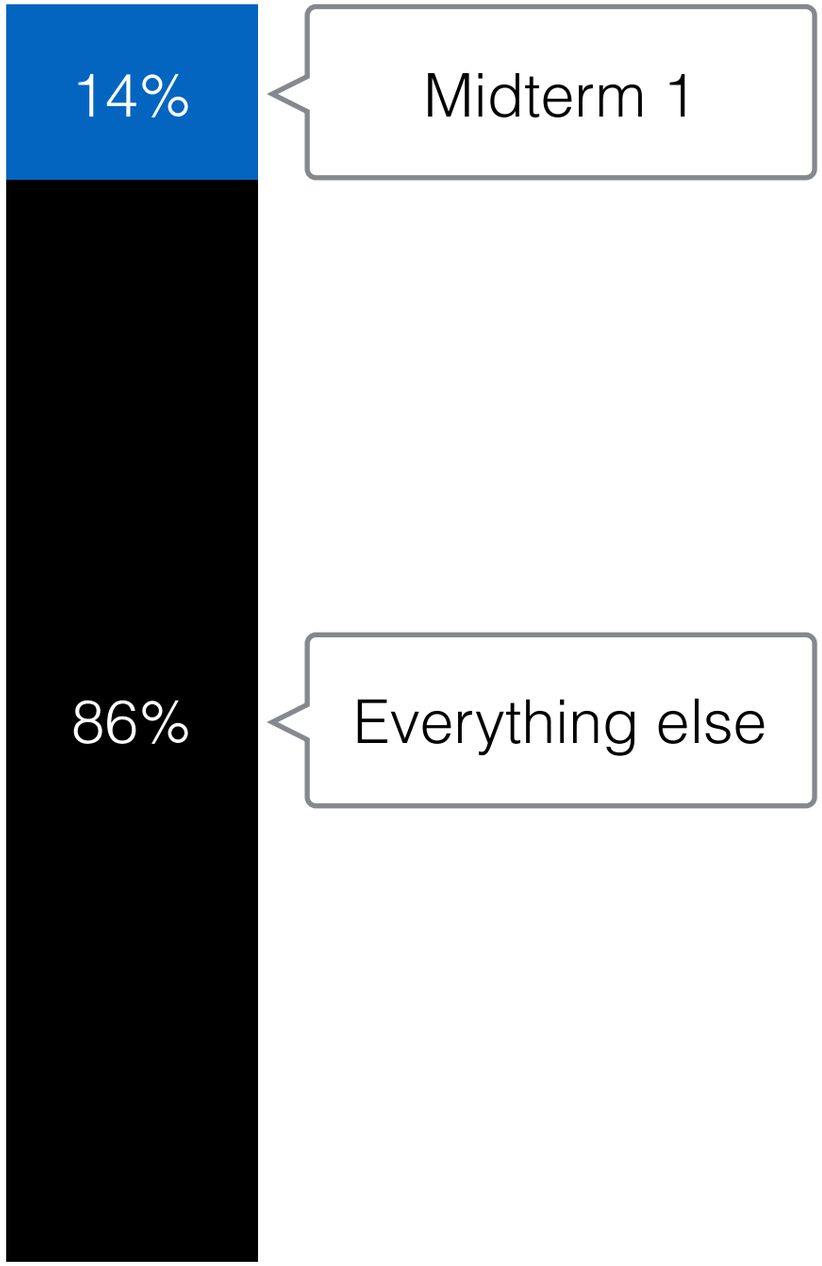
Announcements

- Guerrilla sections Sat 12-3pm in 257 Cory
- Homework 3 released, due 2/14
    - Homework party (check website)
- Midterm on 2/17 7-9pm, fill out alternate form by Sunday
    - Discussion next week is midterm review

Some Perspective



From MT1, Fall 2015



Total Grade

Check Your Understanding

1.

```
[[x for x in range(y)] for y in range(3)]
```

2.

```
def pairs_to_dict(pairs):
    """
    Convert a list of pairs into a dictionary.
    >>> p = [['c', 6], ['s', 1], ['c', 'a']]
    >>> pairs_to_dict(p)
    {'c': 'a', 's': 1}
    """
```

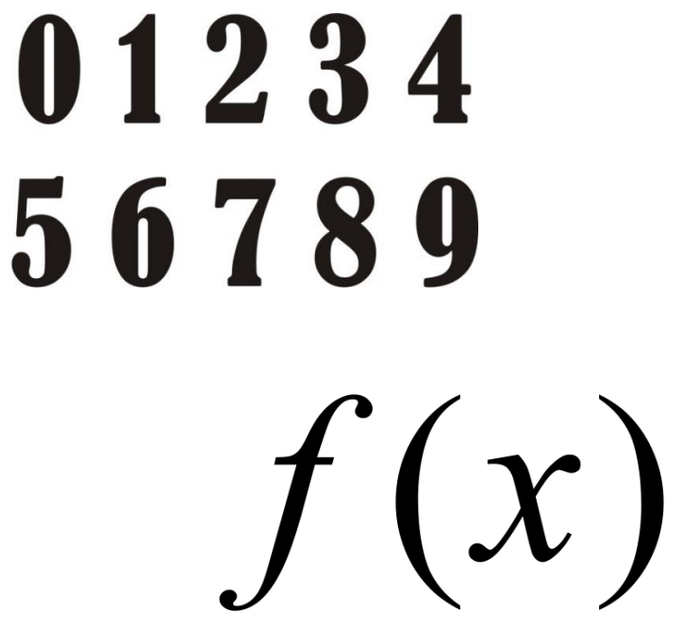
Sequences

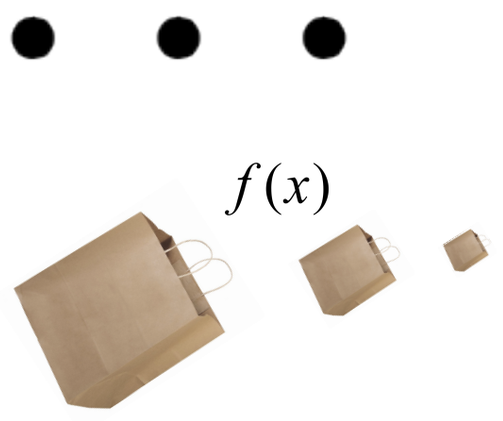
Sequences

Variables (names) generally referred to a single item

A **sequence** is a collection of many items

- Lists: Python's implementation of the abstraction





Lists

Length

Can easily retrieve the length of a list:

```
>>> x = [1, 2, 3]
>>> len(x)
3
>>> y = [x, 4, 5] # Does nesting matter?
>>> len(y)
3
```

Lists

Element Selection

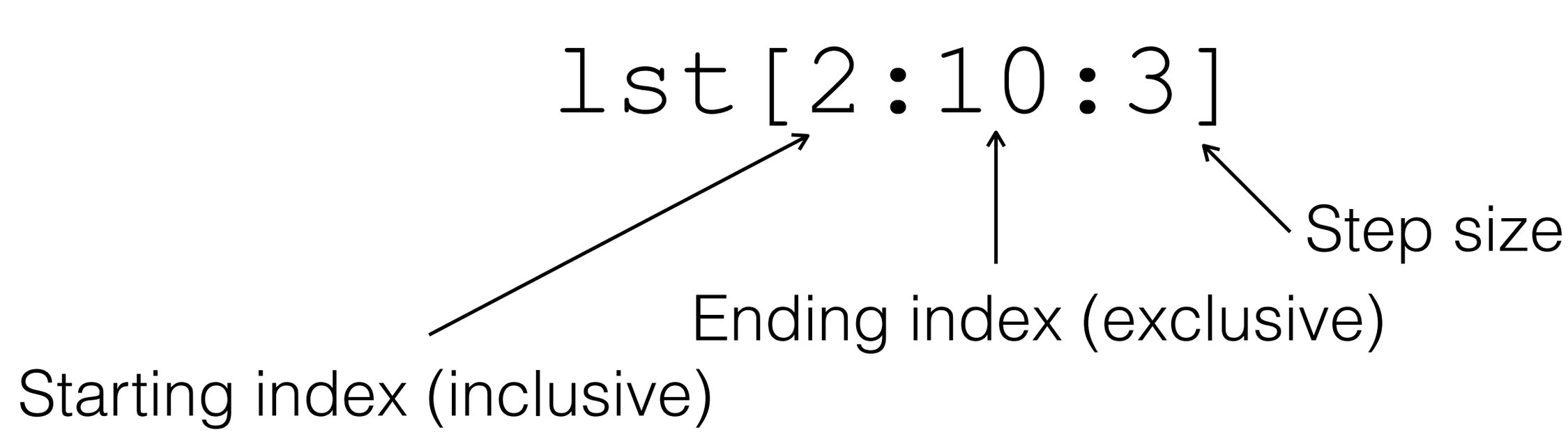
Get an item at an index using bracket notation

```
>>> x = [1, 2, 3]
>>> x[0]
1
>>> x[0] = 10
>>> x
[10, 2, 3]
```

Slicing

Important tool for generating sublists

Anatomy of a slice:



Excluding any part of the slice invokes the default value:
0 for start (positive step), len(lst) for end (positive step), step 1

Lists

Slicing Examples

```
>>> x = [1, 2, 3]
>>> x[0:2]
[1, 2]
>>> x[0:2] == x[:2]
True
>>> x[0:2:-1]
[]
>>> x[2:0:-1]
[3, 2]
```

Lists

Odds & Ends

`for` can be used to loop through lists

```
>>> x = [1, 2, 3]
>>> for elem in x: #elem can be any name
...     print(elem)
1
2
3
```

Lists

Odds & Ends

Check membership using `in`

```
>>> x = [1, 2, 3]
>>> 1 in x
True
>>> “bananas” in x
False
>>> 1 in [x]
False
```

Lists

Odds & Ends

`range` is a useful function that returns a sequence

```
>>> x = range(0, 3) # 0, 1, 2
>>> range(0, 3, 1) == range(3) # Like slicing?
True
>>> for n in x:
...   print(n)
0
1
2
```

Lists Questions

WWPD - Page 2, Q1

```
>>> a = [1, 5, 4, [2, 3], 3]
>>> print(a[0], a[-1])
1 3
>>> len(a)
5
>>> 2 in a
False
>>> 4 in a
True
>>> a[3][0]
2
```

Lists Questions

WWPD - Page 3, Q1

```
>>> a = [3, 1, 4, 2, 5, 3]
>>> a[1::2]
[1, 2, 3]
>>> a[:]
[3, 1, 4, 2, 5, 3]
>>> a[4:2]
[]
>>> a[1:-2]
[1, 4, 2]
>>> a[::-1]
[3, 5, 2, 4, 1, 3]
```

Lists

List Comprehension

Quick way of making lists by applying **expressions** to elements in **another sequence**

```
[<map exp> for <name> in <iter> if <filter>]
>>> [x for x in range(4)]
[0, 1, 2, 3]
>>> [x * 2 for x in range(4) if x % 2 == 1]
[2, 6]
```

Lists Questions

WWPD - Page 4, Q1

```
>>> [i + 1 for i in [1, 2, 3, 4, 5] if i %
2 == 0]
```

[3, 5]

```
>>> [i * i - i for i in [5, -1, 3, -1, 3]
if i > 2]
```

[20, 6, 6]

```
>>>  [[y * 2 for y in [x, x + 1]] for x in
[1, 2, 3, 4]]
```

[[2, 4], [4, 6], [6, 8], [8, 10]]

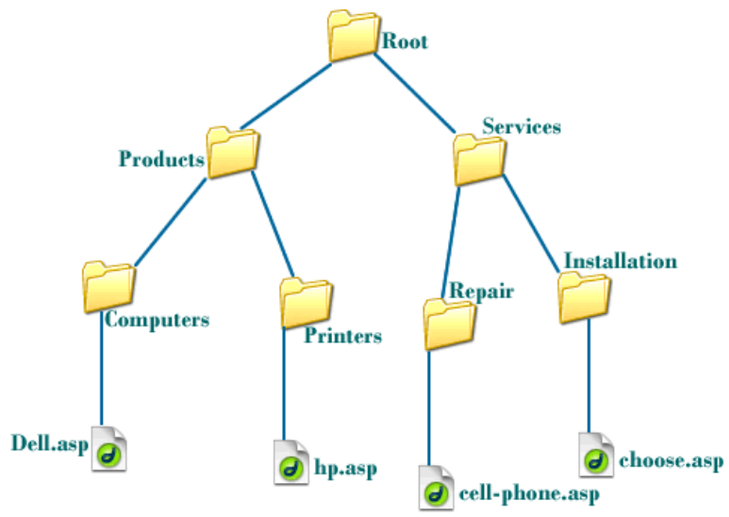
Trees

Trees

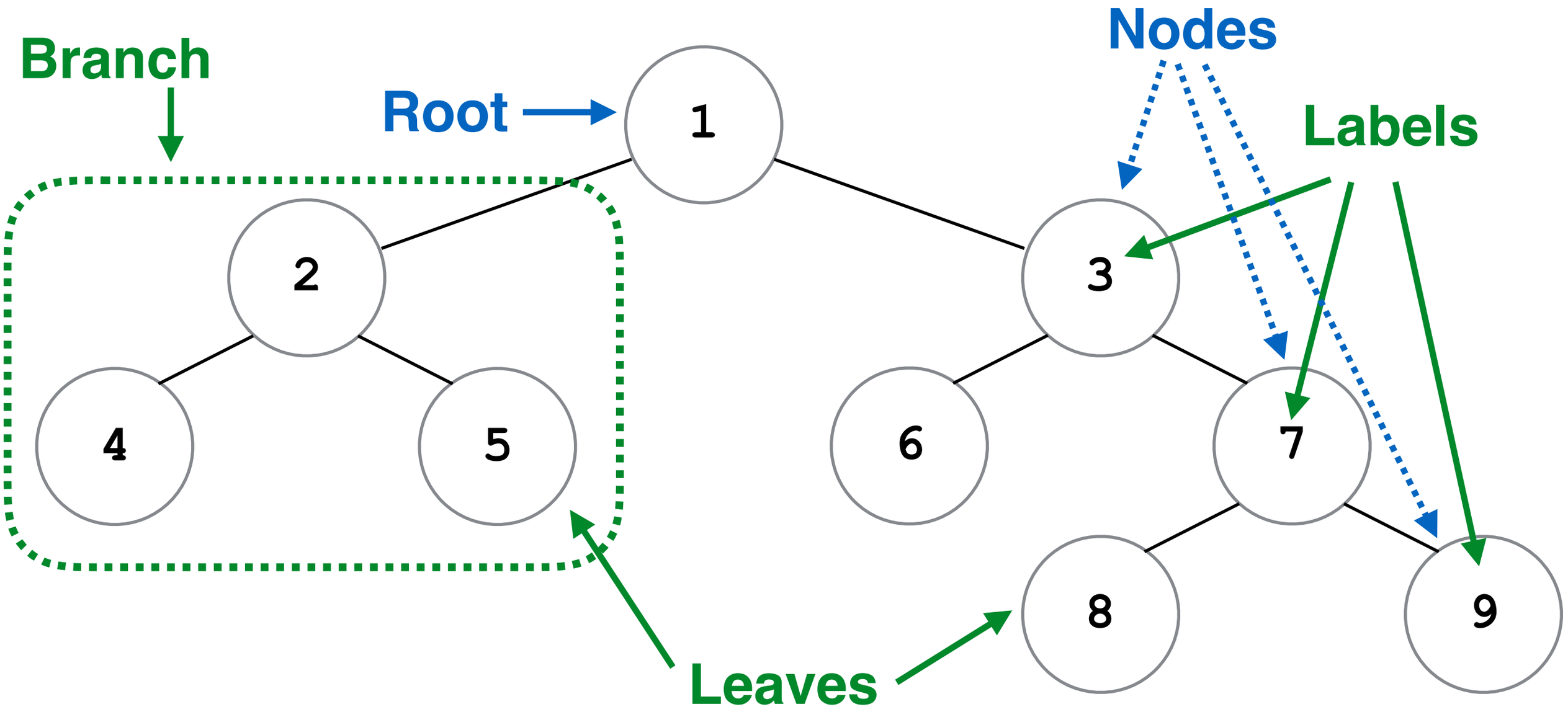
Storing things in order like a list is boring…

In real life, you **see trees everywhere!**

- Taking notes
- Directory structure on your computer
- Nature and stuff, I guess



Trees



Credit: Based on Prof. DeNero's tree diagram [Fa 16 CS 61A]

Trees

Constructor:

```
tree(label, branches=[])
```

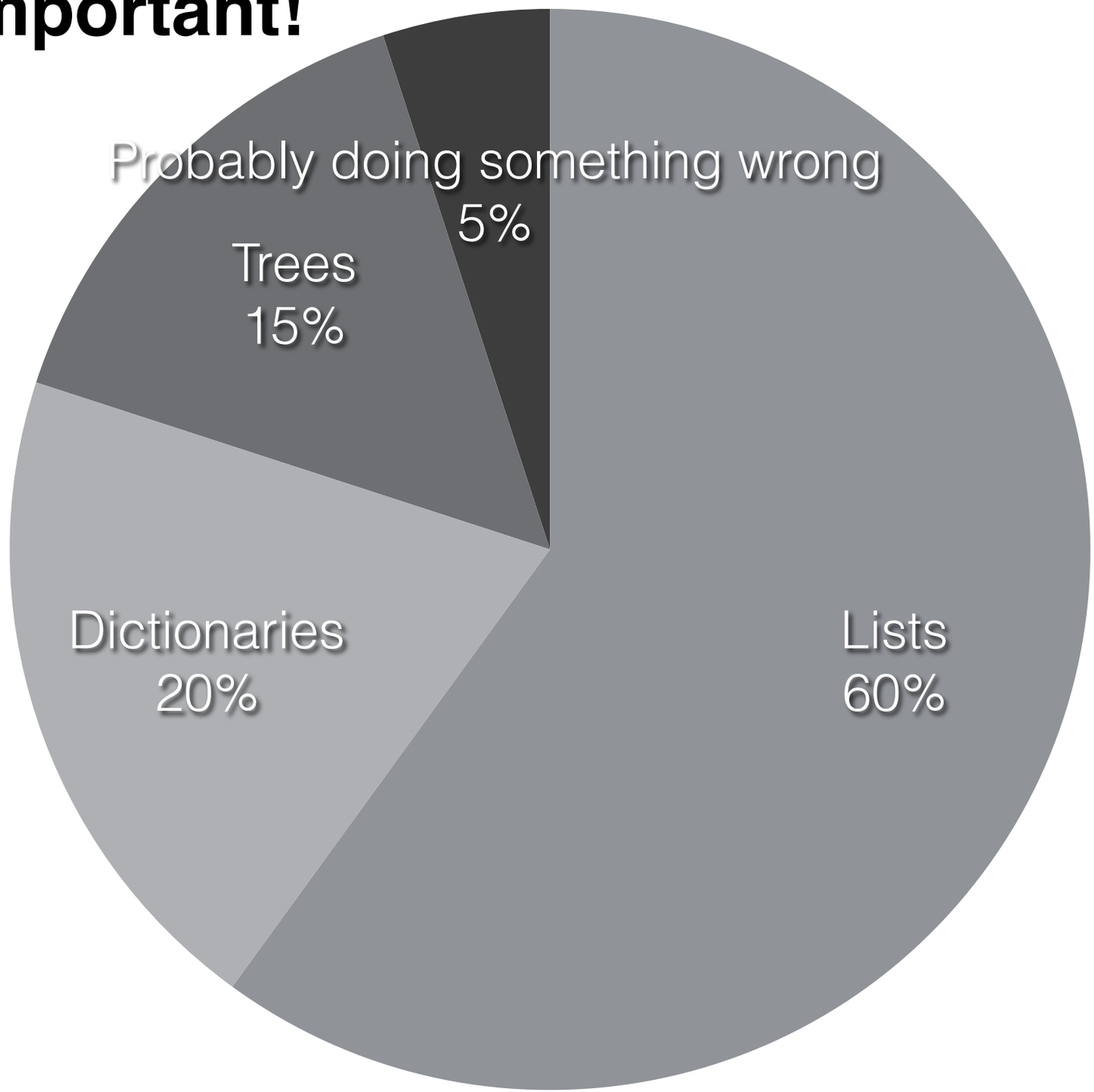
Selectors:

```
label(t), branches(t), is_leaf(t)
```

Why do these matter?

These sequences are important!

Data structures I use:*



*Numbers totally made up (kinda)

Data Abstraction

Focus on **what happens**, not how it happened

- **Abstract data type (ADT)** - represents an object/ thing in code. Abstract since we (as the user) don't need to know how it was built and how it works!

- **Constructor** - creates an ADT

- **Selector** - retrieve information from an ADT

What's the big deal?

I'll just break a data abstraction. What's the worst that could happen?

In all seriousness, consistency is important!