

EECS 12: Lecture 6

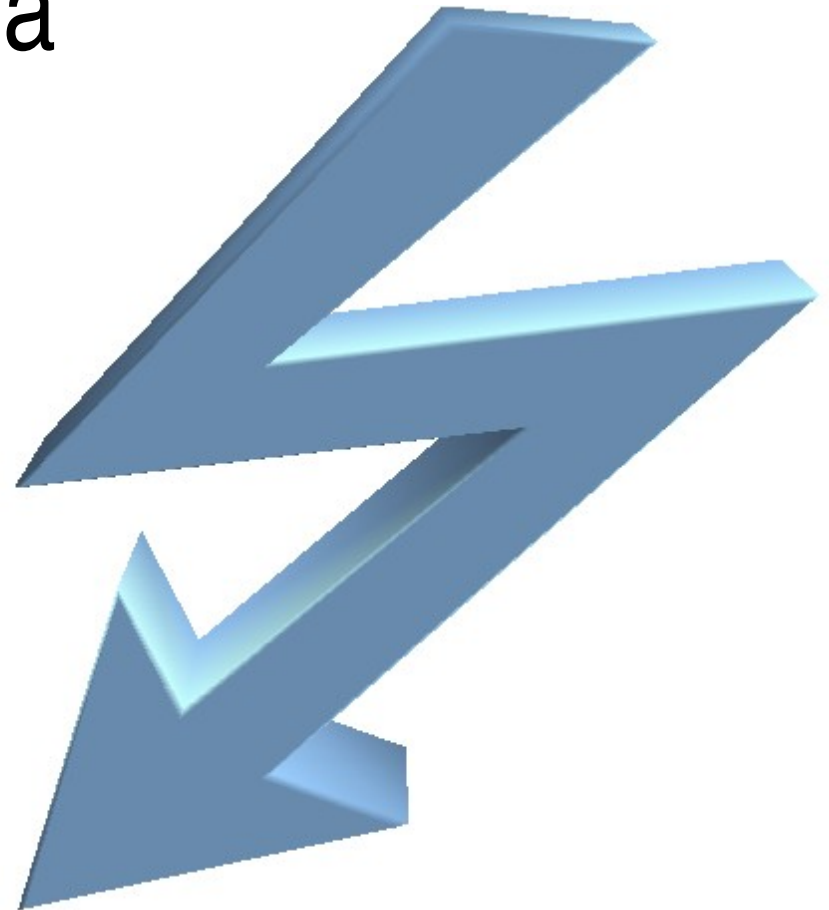
Trees, Files, Exceptions, and OOP

Mark E. Phair
mphair@gmail.com
UC Irvine EECS

July 19th, 2006

Agenda

- Extra Credit Project
- Trees
- OOP Intro: Classes
- Files
- Exceptions
- Python factoid of the day



We apologize for the fault in the agenda.
Those responsible have been sacked.

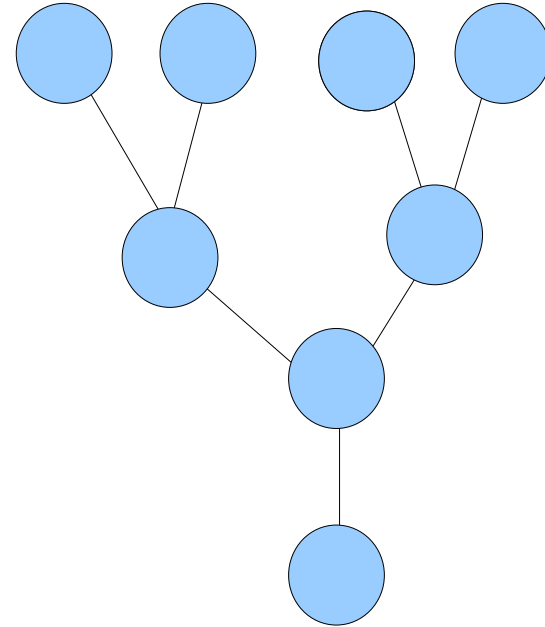
Extra Credit Project

- 50 (Midterm) Bonus Points
- Modify homework 4.1 (data analyzer) so that you can keep track of the operations performed and re-run them on different data
 - Operation: `startproc procname`
 - Operation: `endproc`
 - Operation: `doproc procname`
 - Hint: store procedures in a separate dictionary of lists with *procname* as the key
- Due Friday July 28th at 9AM (NO LATE CREDIT)

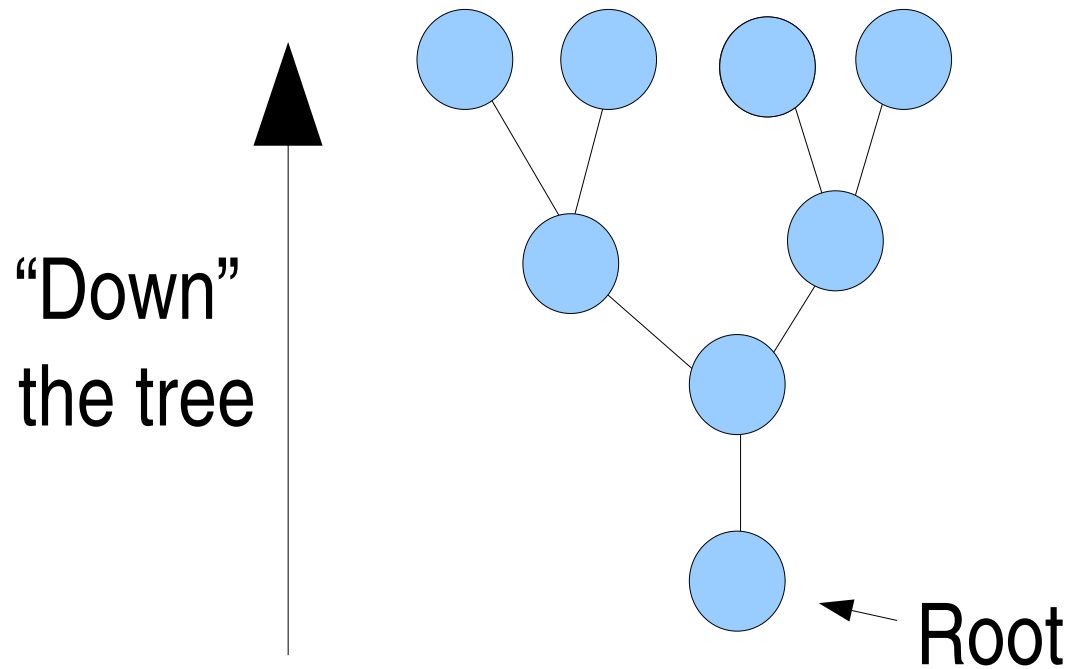
Trees

A tree is a data structure that:

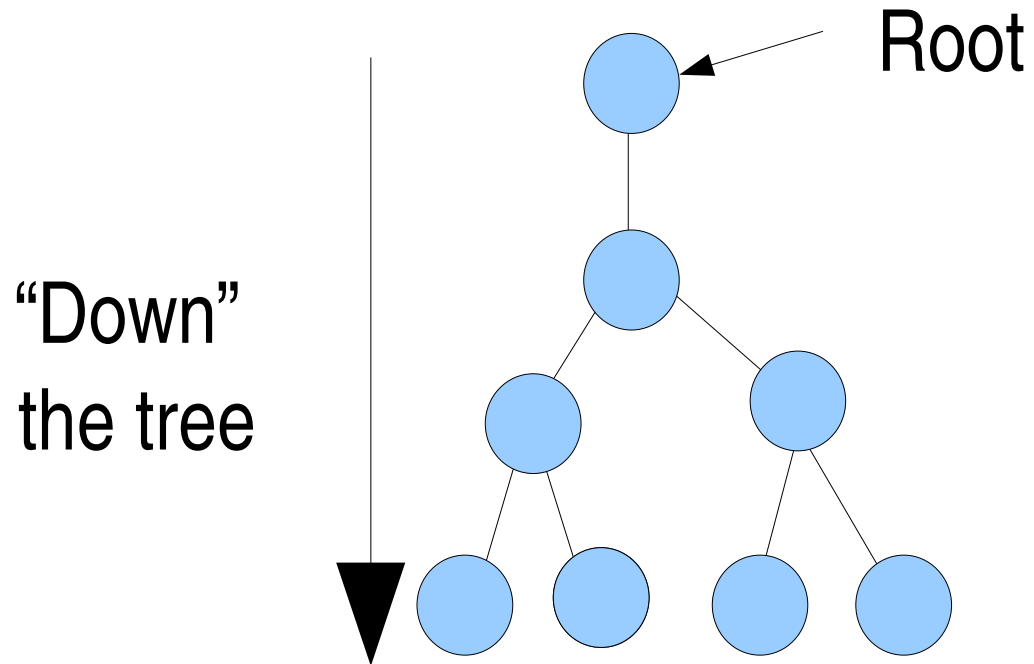
- Has a single root
- Branches
- Does not recombine



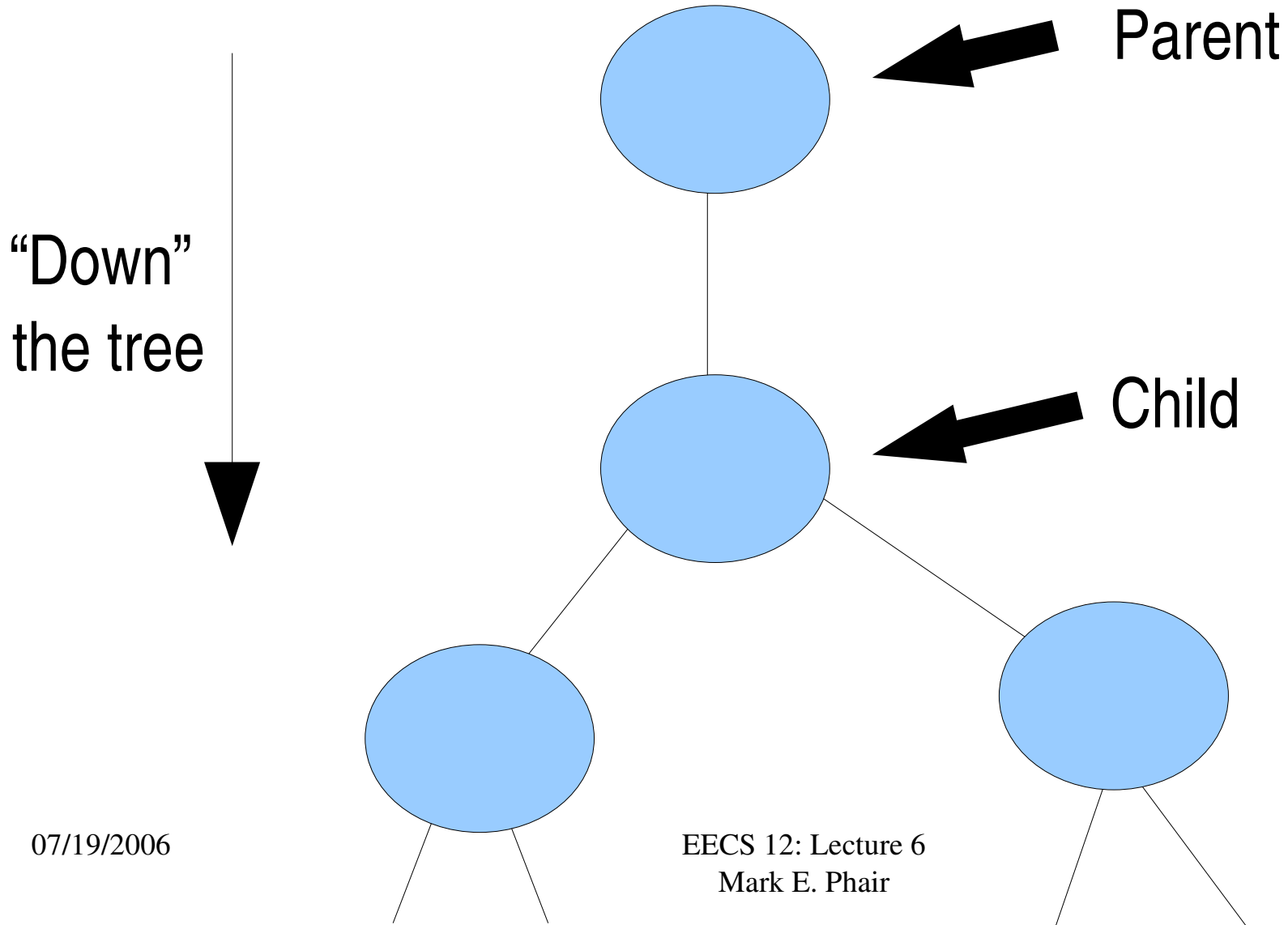
Trees Continued



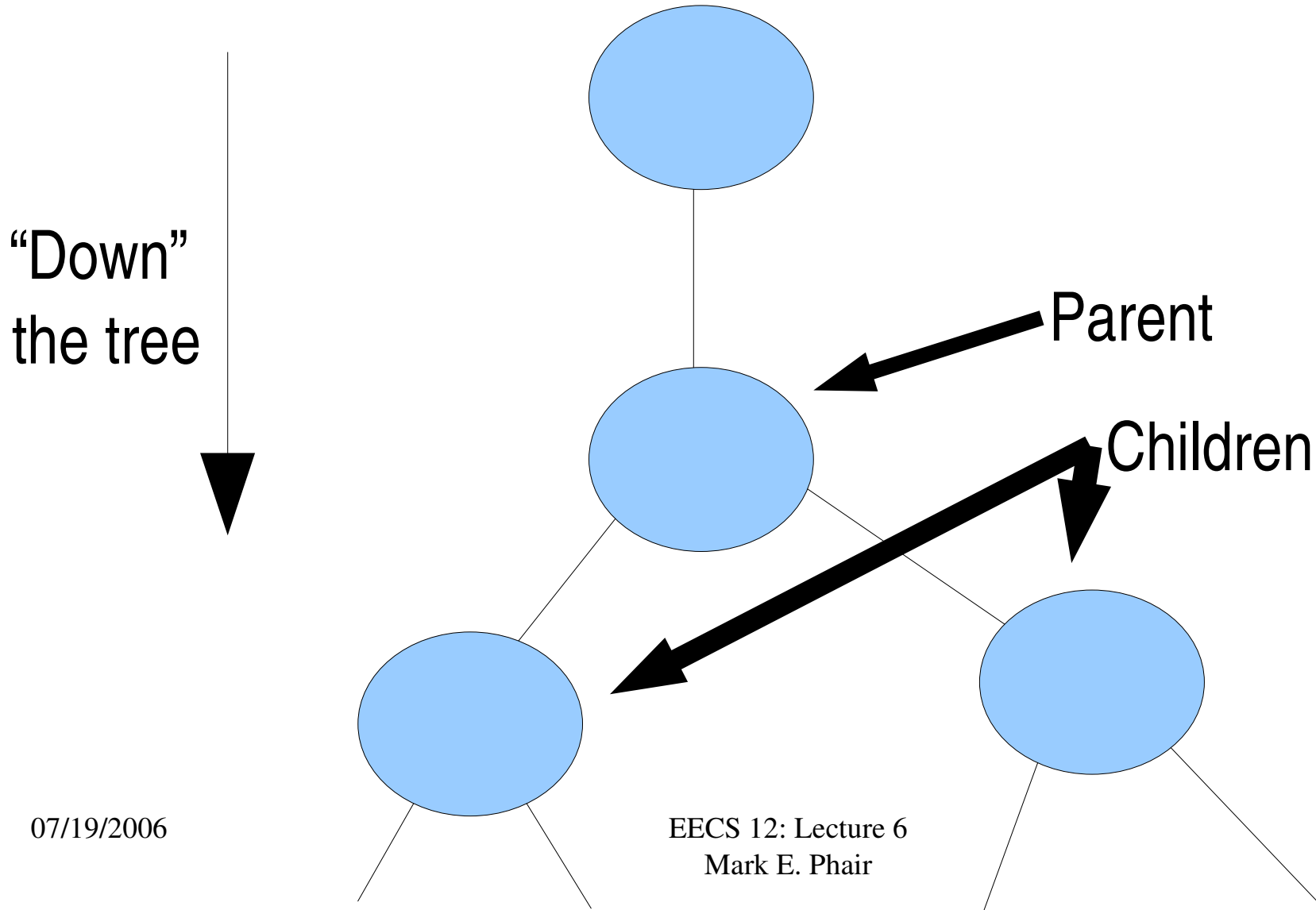
Trees Continued



Trees Continued

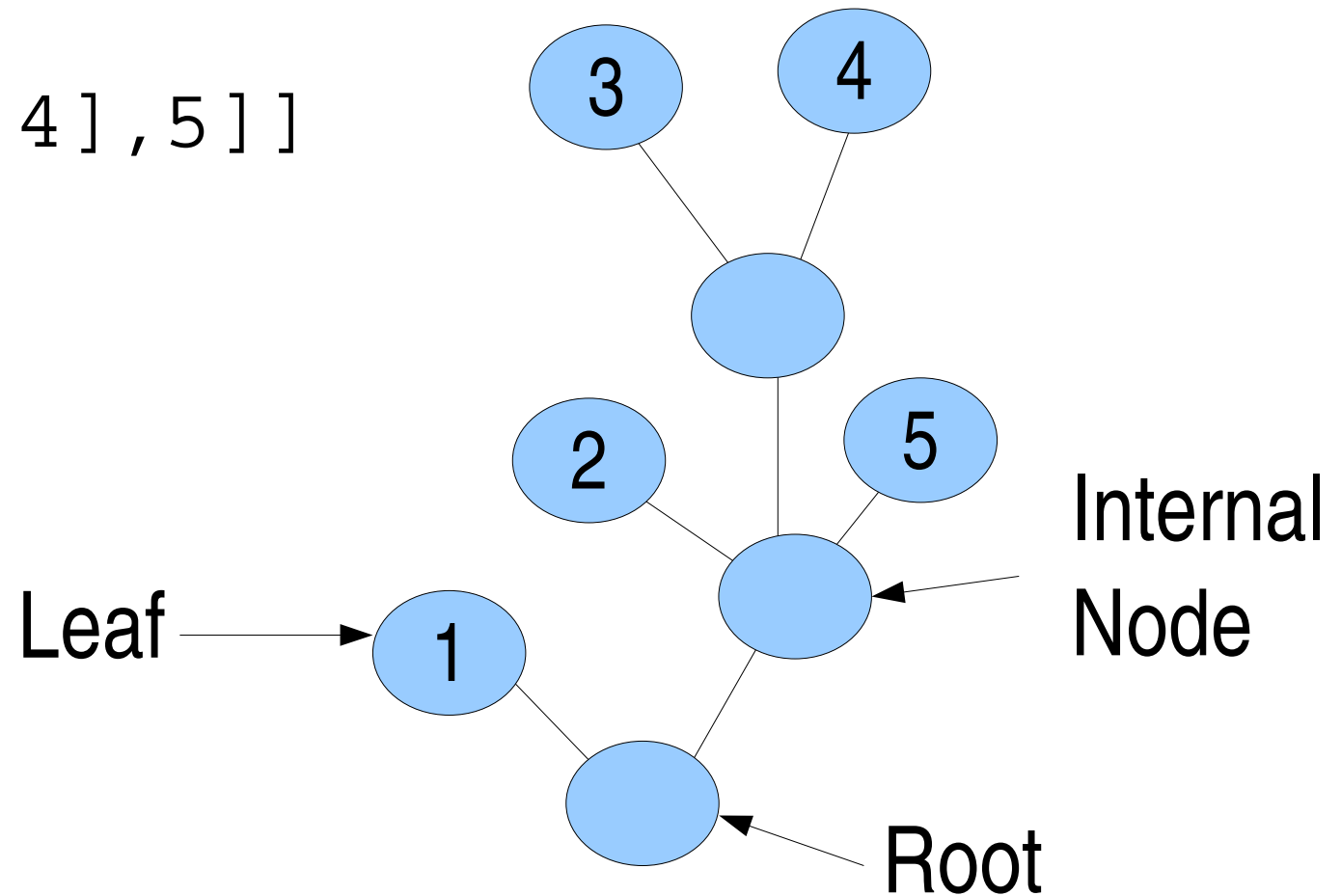


Trees Continued



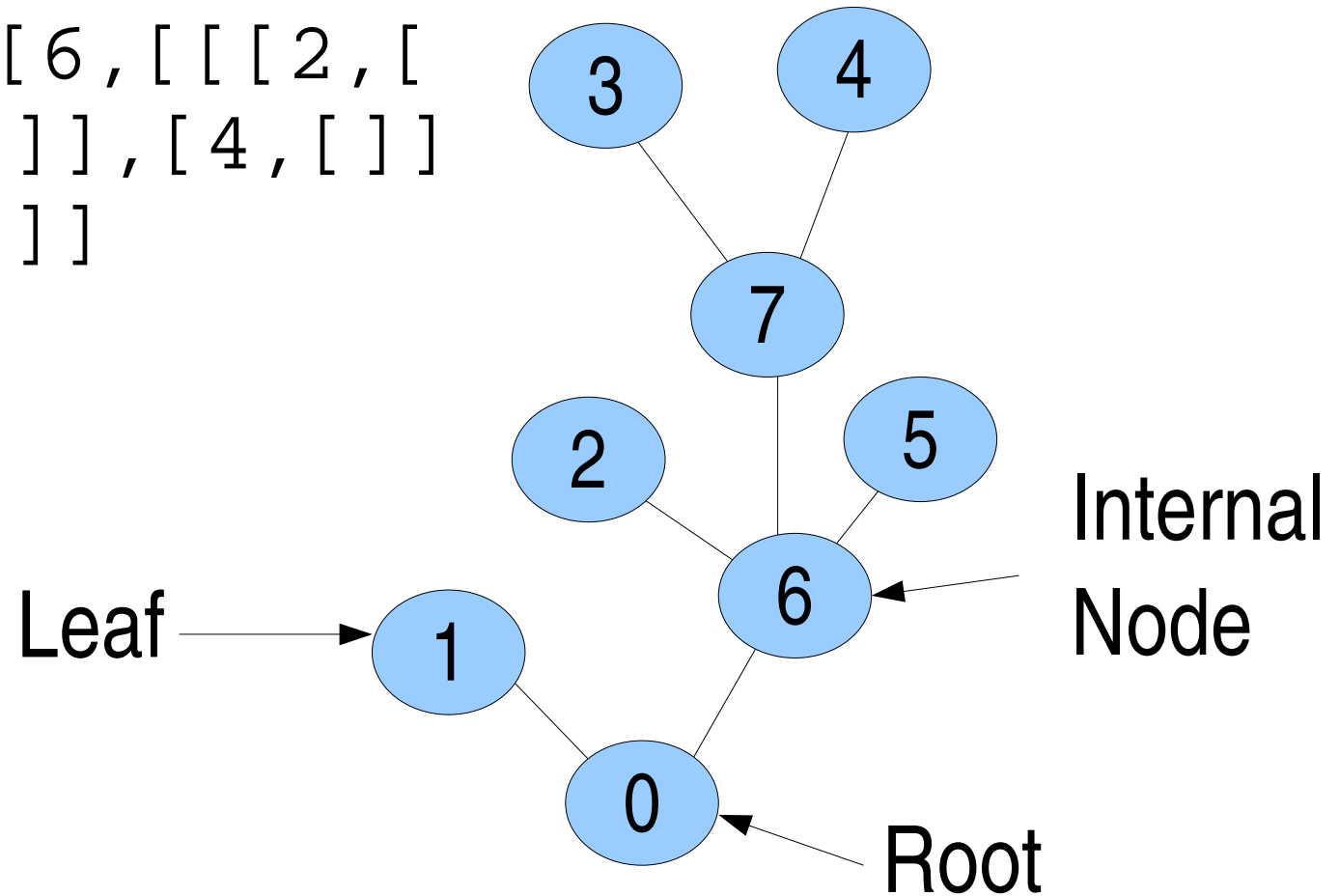
Nested lists can be used to represent trees

`[1, [2, [3, 4], 5]]`



Special Format to name Internal Nodes

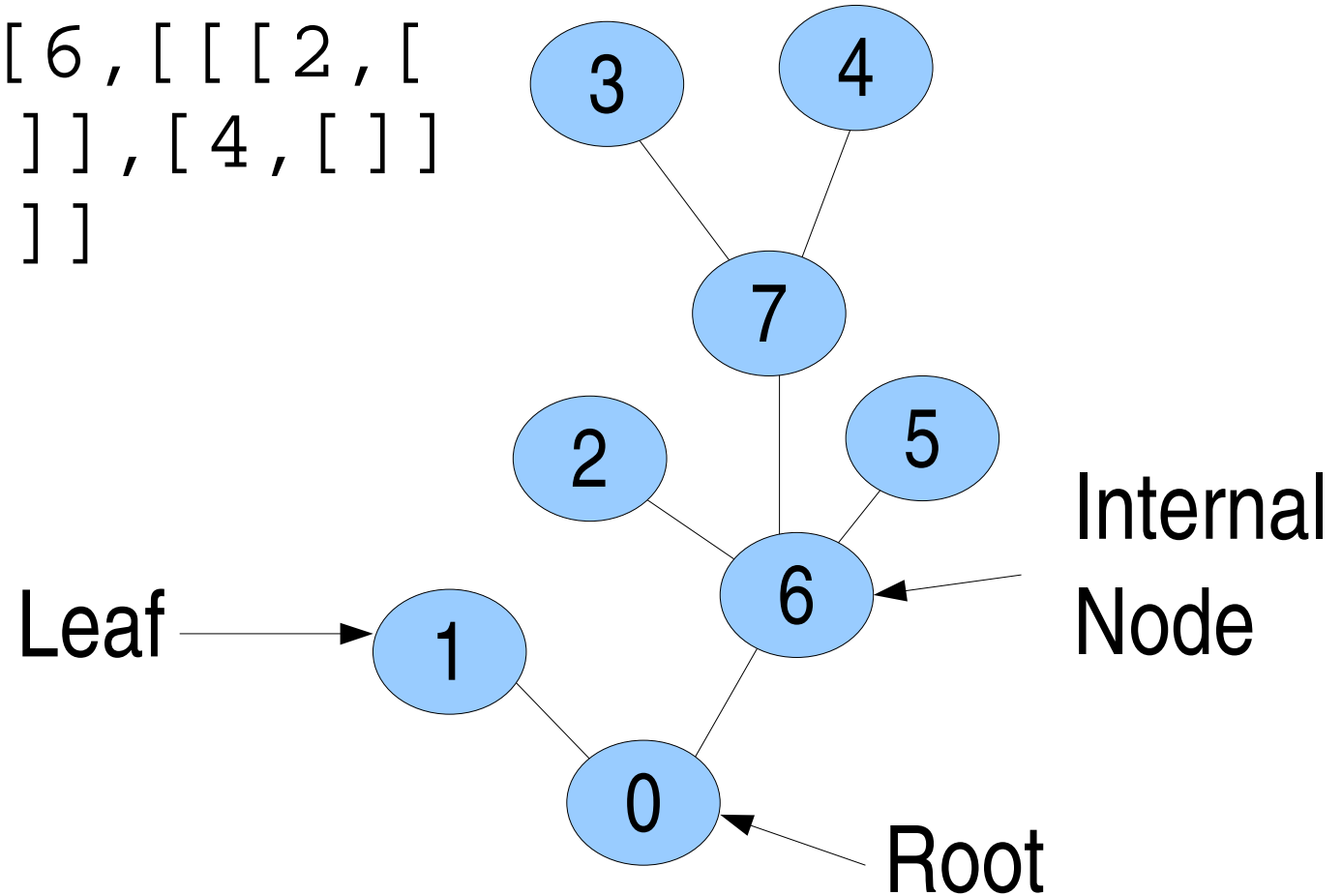
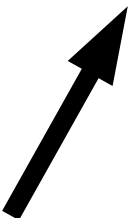
```
[0, [[1, []], [[6, [[[2, [
  ]], [7, [[3, []], [4, []]
  ]], [5, []]]]]]]]
```



Special Format to name Internal Nodes

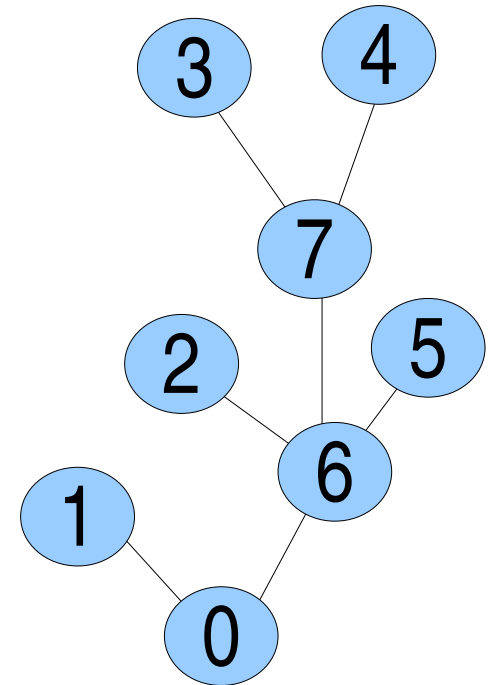
```
[0, [[1, []], [[6, [[[2, [
  ]], [7, [[3, []], [4, []]
  ]], [5, []]]]]]]]
```

Huh?



Another Way: Classes

```
class TreeNode:
    pass # class needs a body
root = TreeNode() #instantiate
root.value = 0    #attribute
root.children = []
root.children.append(TreeNode())
...
```



Class Vocabulary

- A class is a *template* for an *object*
- Classes can have variables as part of them... they are called *attributes*
- An object based on a class is an *instance* of that class
- Creating an object is called *instantiation*, and you *instantiate* the object

Let's Explore

- Create a class called `Frog`
- Instantiate a `Frog` object and store it in the variable `prince`
- Give the `prince` a `home` attribute and assign it the value `'Far far away'`

Files

- You are already familiar with files and directories (folders) from using modern operating systems
- Files are one way that we can *persist state* between runs of the same program
- Files are either text files (like your python programs) or binary files (like executables, PDFs, images)

Directories

- Directories are a tree structure
- Files are the leaves
- On windows
 - the drive letter is the root (C:)
 - the directory character is \ (what about *in* python?)
- UNIX (and Linux)
 - root is /
 - directory character is /

Files in python

- Python can read and write files
- you can open files with `open(filename, mode)`
 - `filename` is a string with the name of the file
 - `mode` is a string: 'r' for read, 'w' for write
- Once you've opened a file you can use `read` and `write`

A Simple (cheating) Quine

- A Quine is a program that prints itself
- It is considered “cheating” to do the following:

```
import sys
name = sys.argv[0] # program name
thisFile = open(name, 'r')
print thisFile.read()
thisFile.close()
```

A Simple (cheating) Quine: with write

This one writes out the result to a file instead of printing

```
import sys
name = sys.argv[0] # program name
thisFile = open(name, 'r')
outFile = open('out.py', 'w')
outFile.write(thisFile.read())
thisFile.close()
outFile.close()
```

Let's Explore Together

Let's Write a program that:

- asks the user for a filename
- takes input from the user until the user types quit
- writes all the input to the file

Exceptions

- Let's allow the user to use Ctrl-C instead of typing quit
- Usually, Ctrl-C will give an error, called `KeyboardInterrupt`
- We can stop that error from reaching the user with a `try . . . except` block

KeyboardInterrupt Exception

```
try:  
    raw_input( '?' )  
except KeyboardInterrupt:  
    print 'You pressed Ctrl-C'
```

Let's `try` it...

Rewrite the program we wrote earlier, but use `try...except` instead of checking for quit.

Use `try . . . except` to catch anything!

```
>>> def a(b): return b
```

```
>>> a(2,3)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in ?
```

```
TypeError: a() takes exactly 1  
argument (2 given)
```

Catching TypeError

```
>>> def a(b): return b
>>> try:
...     a(2,3)
... except TypeError:
...     print 'Wrong Number of args'
...
```

Wrong Number of args

Can catch different types of errors

```
try:
    raw_input( '? ' )
    raw_input( '?', 1, 2, 3, 4, 5, 6, 7 )
except TypeError:
    print 'Wrong number of arguments'
except KeyboardInterrupt:
    print 'You pressed Ctrl-C'
```

Python Factoid of the day: Pickle 'em up

The pickle module provides a way to store most python objects

```
>>> import pickle
>>> f = open('out', 'w')
>>> pickle.dump([1,2,3], f)
>>> f.close()
>>> f2 = open('out', 'r')
>>> lst = pickle.load(f2)
>>> lst
[1, 2, 3]
```