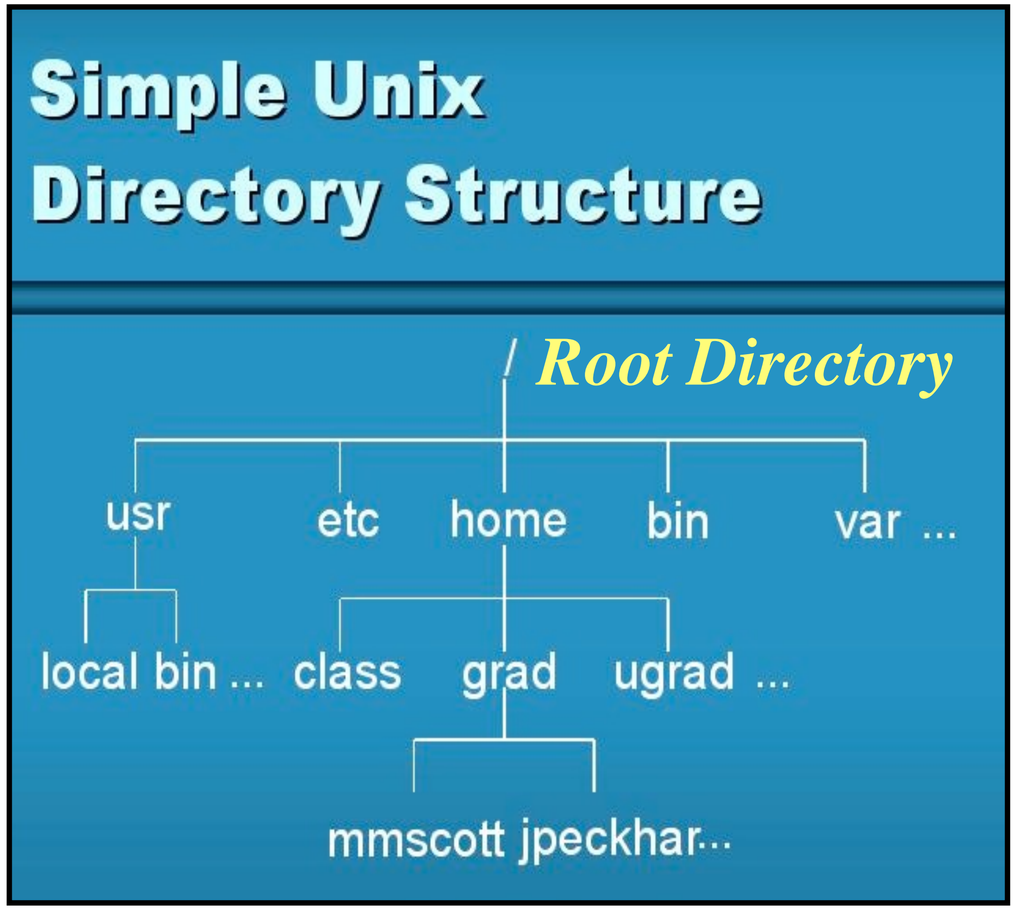
Introduction to Unix and Python Programming

- Working with Unix
  - Basic Commands

- Python Programming
  - basic steps to create and run a python program
  - the basic elements of most python programs
  - filenames and variable names
  - variable types
  - debugging a python program

A Few Points about File Path

1. Unix file system is organized using a *tree* of connected directories and subdirectories
   - ***i.e., a directory tree***
2. Each directory is given a fully spelled out name *or some kind of abbreviated name*
3. The directory at the base of the *tree* is given a very special name (*and symbol*) called the ***root directory*** and **instead of a name it is designated by: / (*i.e., a forward slash*)**



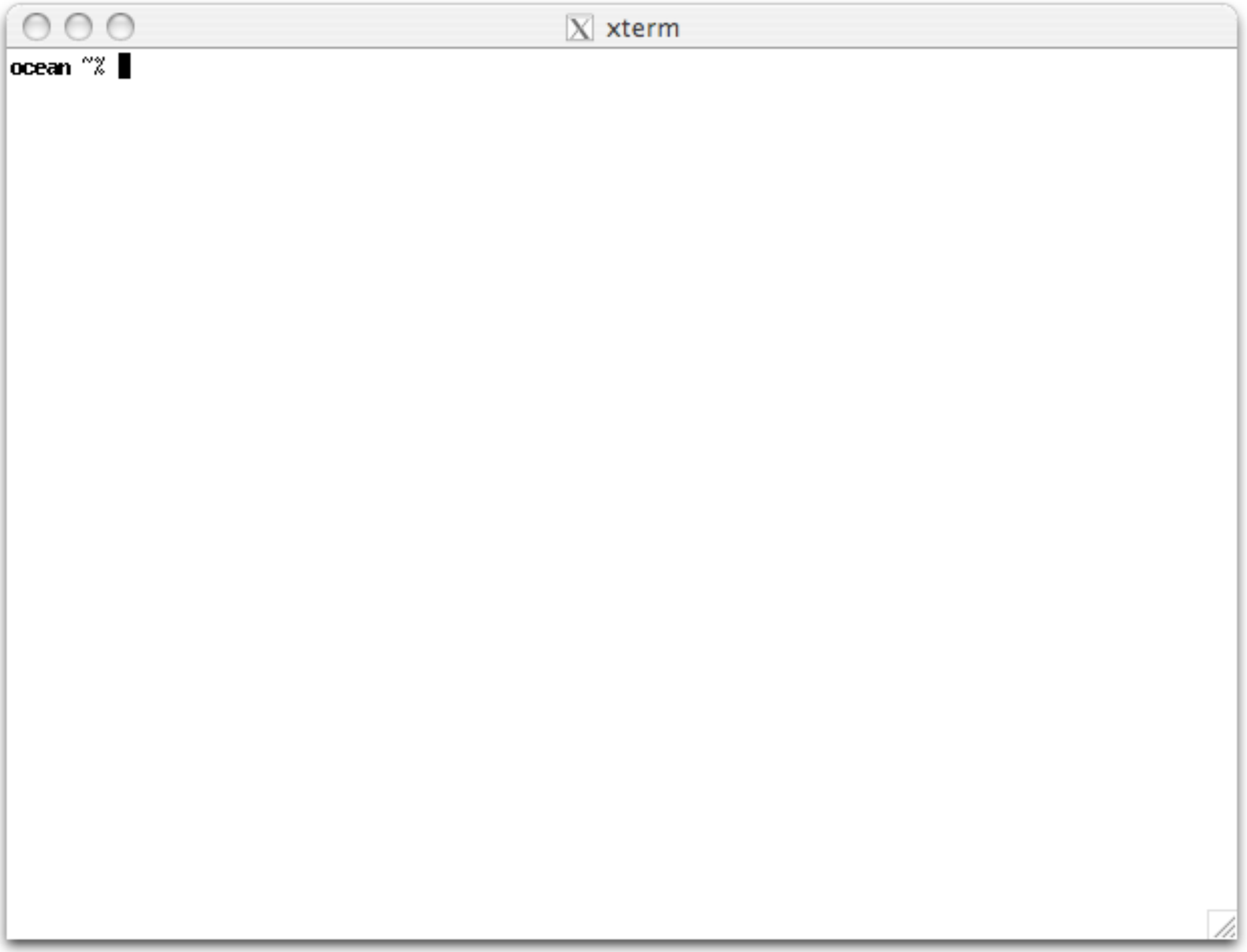
4. **Unfortunately**, the forward slash **is also used** to denote ***the separation between a directory and a subdirectory.***
5. ***A Directory Path*** spells out where a given file is located within the directory tree.
   - for example: **/dir1/dir2/dir3/filename.txt**

UNIX Basics

1. When you open a new Unix terminal window, you automatically reside in your ***home directory.*** *This directory is* often denoted with the unix shorthand symbol: **~/**
   So: **~/** is equivalent to **/Users/bcm3/**
2. **NOTE:** ***Python*** ***does not*** recognize the *Unix* shorthand notation of **~/** So in your python programs you need to **explicitly spell out the path of your home directory**.

   In stead of this: ~/ You should always use this: *~/Users/Your-User-Name/*
3. All files and directories below your **home directory** are usually **owned** by you.
4. The directory where you currently reside is called your **Working Directory** You can *print* the name of the working directory by typing the unix command: ***pwd***
5. You change from your current working directory to another directory with the change directory command: ***cd***
6. You will often want to return to your Home Directory. Unix has made this easy for you. Simply type: ***cd*** **and then <return>**

The UNIX Command Line



Basic Unix Commands

1.	***pwd***	***p***rint ***w***orking ***d***irectory path
2.	***cd*** dir1	***c***hange to ***d***irectory dir1
3.	***ls***	***l***i***s***t file names (and subdirectories) contained in the current working directory
4.	***cp*** fname1 fname2	make a ***c***o***p***y of fname1 and call it fname2
5.	***mv*** fname1 path/	***m***o***v***e fname1 to a new directory described by path/
6.	***cp -r*** dir1 path/	***c***o***p***y ***r***ecursively dir1 (*and all contained subdirectories*) and place it in a new directory described by path/
7.	***mkdir*** dir	***m***a***k***e a new ***dir***ectory
8.	***rm*** fname	***r***e***m***ove (delete) fname
9.	***rmdir*** dir1	***r***e***m***ove empty ***dir***ectory
10.	***rm*** -r dir1	***r***e***m***ove (***r***ecursively) dir1 and its entire content

1. **full file name** = **directory path** + **base name** e.g., **/dir1/subdir2/subdir3/fname.txt**
2. **Unix distinguishes between upper case and lower case letters**

The directory path for a given file can be referenced **two different ways**:

Absolute Path: the path begins with **the root directory (/)**

Relative Path: the path begins with **the current working directory you are in**

For Example: If you are currently in your home directory then **the following two expressions are equivalent.** Notice the relative path does not have the **/** root directory at the beginning of the path name.

- **/Users/Your_Username/data/fname.dat**
- **data/fname.dat**

Another Example: If you are in your home directory and want to change directories to the ***data*** directory that is in your home directory, then following two expressions are equivalent…

- **cd /Users/Your_Your_Username/data**
- **cd data**

What is Python?

From the Python Homepage:

1. Python is an interpreted, object-oriented, high-level programming language.
2. The high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for
   - For **rapid application development**
   - For use as **a scripting language** to connect existing components

Python is "Open Source"

1. The Basic Python that is found on most computers has a limited set of features
2. But tons of people and organizations have created an extensive set of additional functional capabilities that you can ***Import*** into your program to create a truly powerful data analysis tool.

Python Programming: Basic Steps

1. Write lines of code using a text editor
2. Interpret the lines of code with the python interpreter and then execute the interpreted lines of code
3. Debug program by modifying the lines of code in the text file
4. Repeat steps 1 - 3 as needed

Two Approaches to Using Python

Interactive Mode *and* **Text File Interpreter Mode**

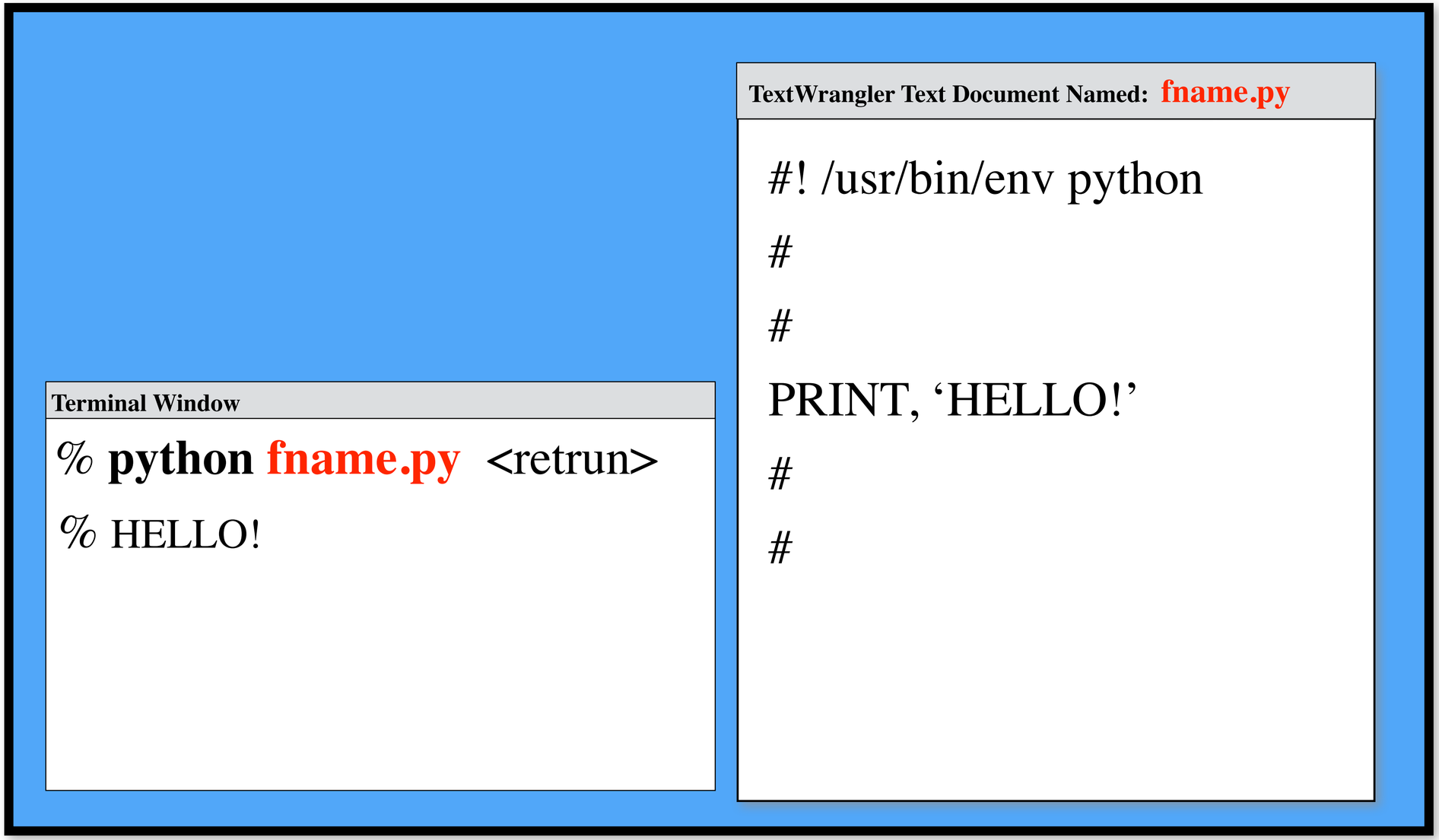
Interactive Mode

1. Open a Unix Terminal Window and then type: **python**
2. Start typing out python commands at the python prompt: **>>>**
3. This approach is really great for quickly checking out how a new python function works.

Text File Interpreter Mode

4. Open a text file with a text editor
5. Write lines of python code into the open text window
6. Save the text file and run the python code contained in the text file typing the following in a Unix Terminal Window: **python textfile.py**
7. This approach is great for writing elaborate programs that you want to use again and again.

Python programming in Interpreter Mode requires **two** windows to be open: 1. Text Document Window and 2. Terminal Window to compile and run the python commands in the text document



Basic Elements of most Python Programs Used in this Class…

#! /usr/bin/env python *(optional)*

1. read data from the hard drive into variable names

2. do basic math with variable names

3. image the data results

4. write data results to a file on the hard drive

The difference between the **File Name** and the **actual Data** contained within the file

```
filename = '/dir_path/S1998148172338_chlor_a.f999x999'
f = open(filename)
actual_data = fromfile(f, dtype=float32)
f.close()
```

1. **filename** is really just a bunch of ASCII text characters that tells the computer where on the hard drive the actually data is stored.
2. **f=open(filename)** assigns the file name to a **logical unit** (called *f* here, but you can call it anything you like e.g. *f1* or *file_1* or *abc*) that is used by a computer for reading/writing from/to the hard drive. The statement also reserves space in RAM memory on the computer chip in preparation for moving (reading) the data on the hard drive into the RAM memory on the chip.
3. **actual_data=fromfile(…)** does the actual reading from the hard drive into RAM memory and is then represented in the program code with the *variable name* **actual_data**
   note: I could have used any variable name besides **actual_data** - e.g. **my_chlor_data**)

Variable Types

- Scalar - single number
- Vector - 1D series of numbers
- Matrix - 2D (or more) set of numbers

- **Floating Point** $+/-1.0 \times 10^{37}$
- **Integer** +/-32767
- **Byte** 0-255
- **String** (e.g., A, B, C…Dog, Cat)

Debugging Python Programs

1. Syntax Errors
   - occur during compiling
   - explanation(s) and line number(s) printed to the Unix window
   - usually a matter of adding a comma or another parenthesis to the offending line

2. Program Errors
   - occur while program is running (progressing down your program code) and causes it to halt
   - a brief, sometimes cryptic, explanation and line number is printed to the unix window
   - these errors are more difficult to track down
   - **read the error message in the unix window carefully!**
     - **use lots of print statements** (or imaging statements for 2D data) **above the position in the code where the error occurred** and moving progressively up the code. The goal is to find the place where the values or dimensions of the program variables are not what you thought they should be.

3. Logic Errors
   - Compare results with what is physically reasonable!!

Debugging Python Programs

- Carefully read the error message printed to the terminal window
  - The error message can be a little cryptic, but try to decipher the basic features…
  - In particular note **the line number** where error occurred!

- There are too many types of errors to mention here, but the general approach is to work your way up the program above the line of code where the error occurred.
  - Print Statements (and Imaging Statements for 2-D problems) are your friend and you should use them a lot to query the values (*and array dimensions*) of certain parameters to see if they are what you expected.