

INTRODUCTION TO PYTHON: DAY ONE

STEPHANIE SPIELMAN

BIG DATA IN BIOLOGY SUMMER SCHOOL, 2015

CENTER FOR COMPUTATIONAL BIOLOGY AND BIOINFORMATICS

UNIVERSITY OF TEXAS AT AUSTIN

TOPICS WE'LL COVER:

Day One, May 26th

- UNIX / Bash
- Python data structures and basic operations

Day Two, May 27th

- Control flow in Python (if, for, while)

Day Three, May 28th

- Writing functions in Python
- File input/output

Day Four, May 29th

- Python modules
- BioPython: Handling sequence data
- Scripting

WHAT YOU WILL GAIN FROM THIS COURSE

- **You will NOT learn Python (in just 4 afternoons)**

WHAT YOU WILL GAIN FROM THIS COURSE

- You will NOT learn Python (in just 4 afternoons)
- The idea is to give you the *resources* and the *skills* to continue learning and applying these concepts on your own

WHAT YOU WILL GAIN FROM THIS COURSE

- You will NOT learn Python (in just 4 afternoons)
- The idea is to give you the *resources* and the *skills* to continue learning and applying these concepts on your own
- ...Ok, you'll learn some Python 😊

WHY LEARN COMPUTER PROGRAMMING?

- **Speed**
- **Automation**
- **Repeatability**

WHY LEARN PYTHON?

- **Higher-level language with extensive functionality**
 - Well-documented
 - Widely-used
 - Very readable and user-friendly
 - Excellent for handling text and files
- **The main drawback is speed**

COMPUTERS ARE STUPID

No, really.



LET'S BEGIN WITH UNIX

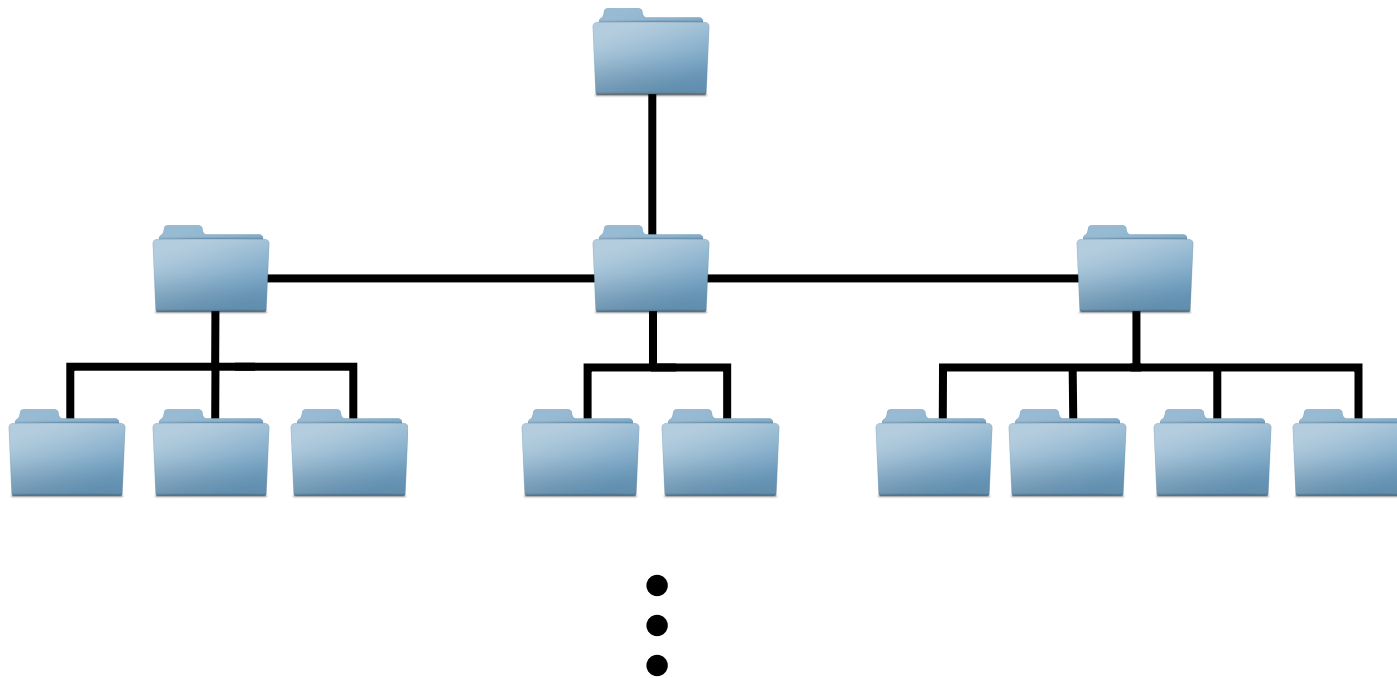
- **UNIX is a computer operating system**
 - Mac and Linux are built on UNIX, but PCs are not 😞

LET'S BEGIN WITH UNIX

- **UNIX is a computer operating system**
 - Mac and Linux are built on UNIX, but PCs are not ☹️
- **We interact with this system using a *shell*.**
 - We'll use Bash (bourne-again shell)

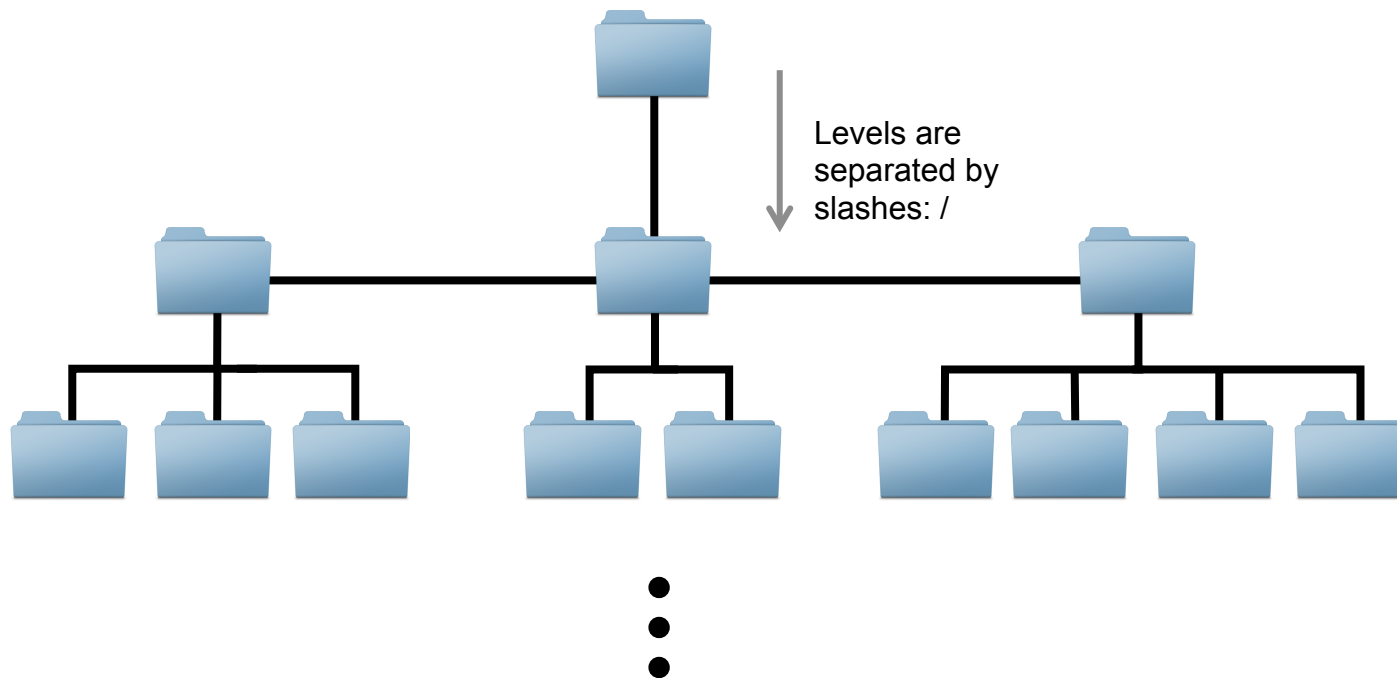
NAVIGATING UNIX SYSTEMS

Files and directories in UNIX systems are organized *hierarchically*



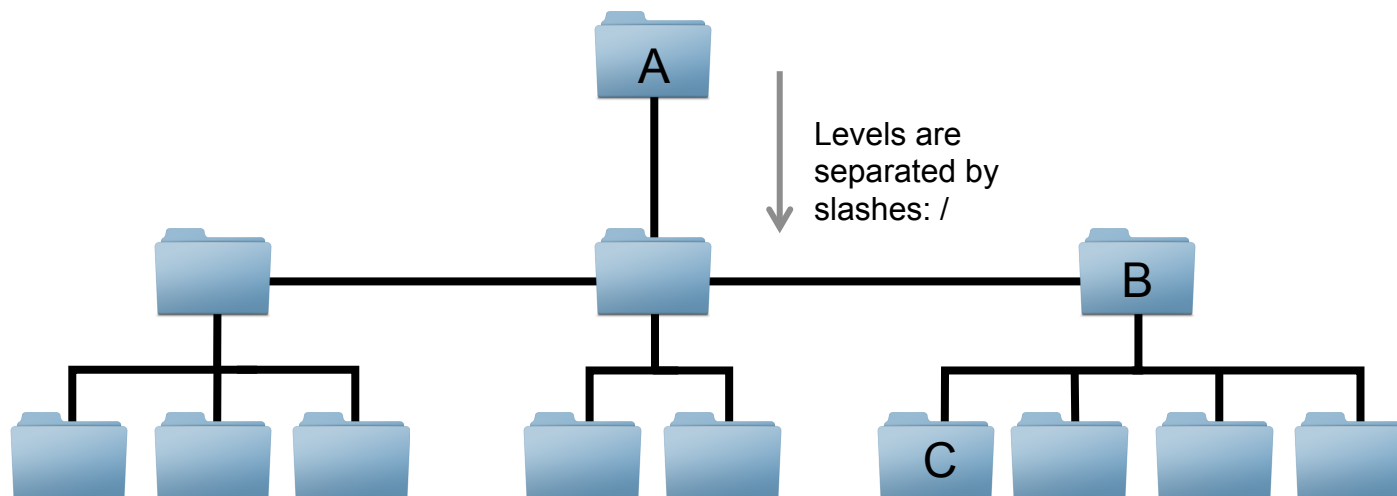
NAVIGATING UNIX SYSTEMS

Every file/directory has a specific address, or **path**, in the hierarchy



NAVIGATING UNIX SYSTEMS

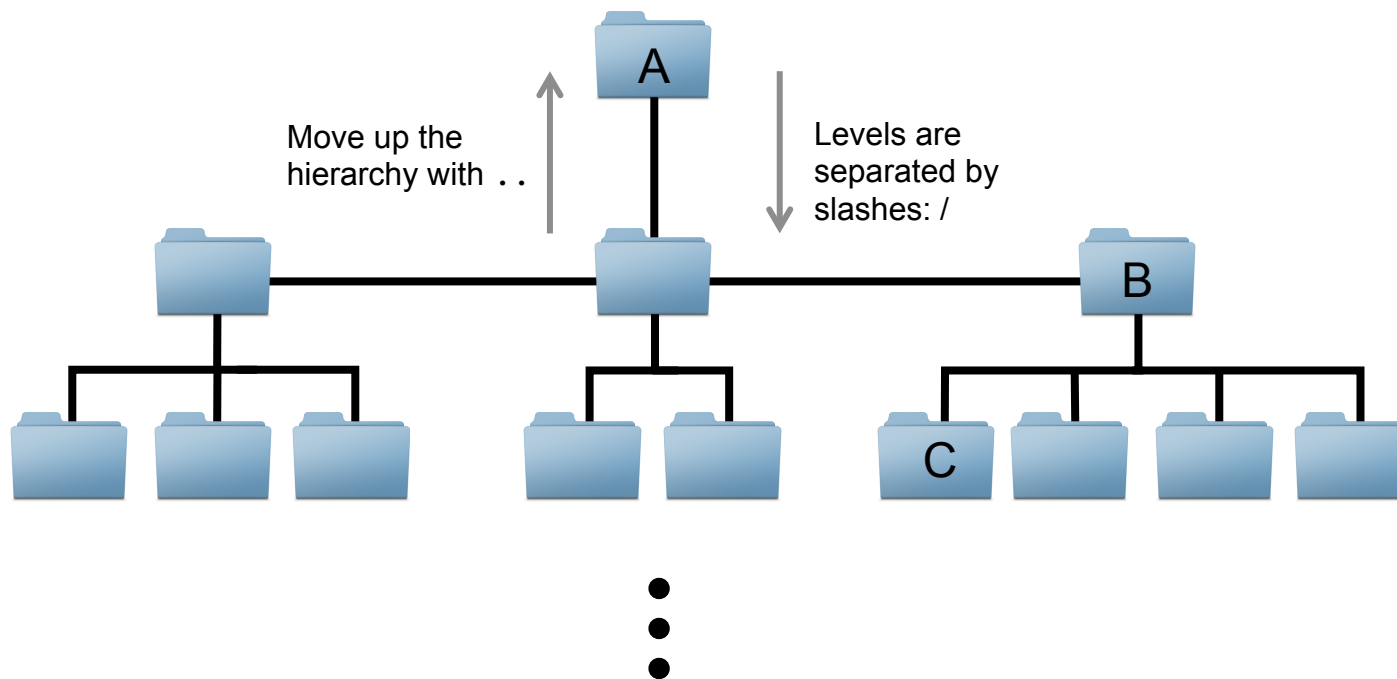
Every file/directory has a specific address, or **path**, in the hierarchy



• The path from A to C is B/C

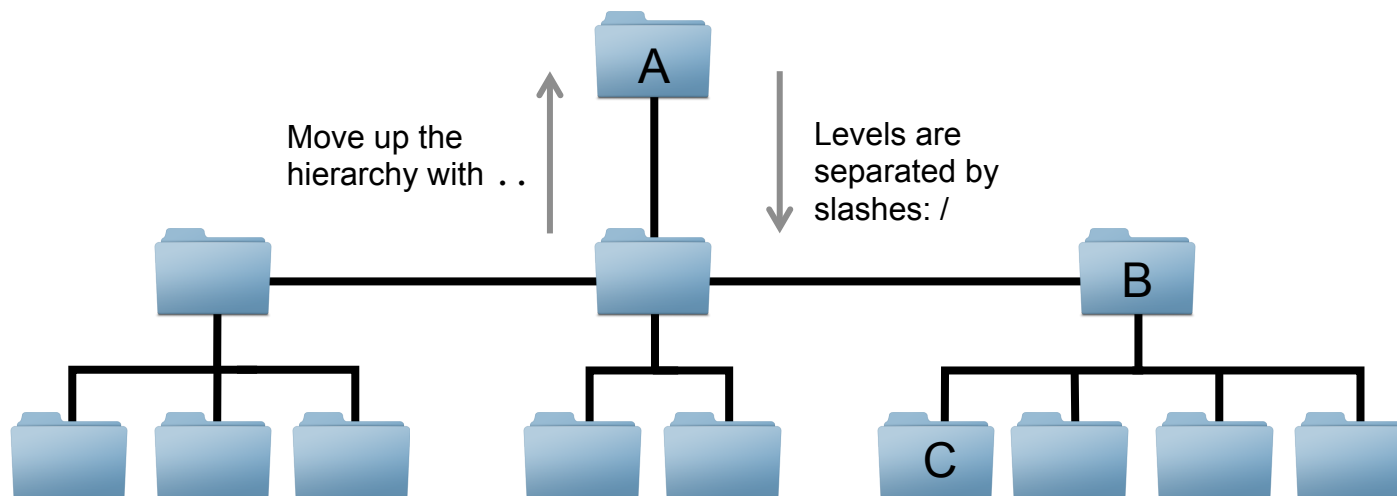
NAVIGATING UNIX SYSTEMS

Every file/directory has a specific address, or **path**, in the hierarchy



NAVIGATING UNIX SYSTEMS

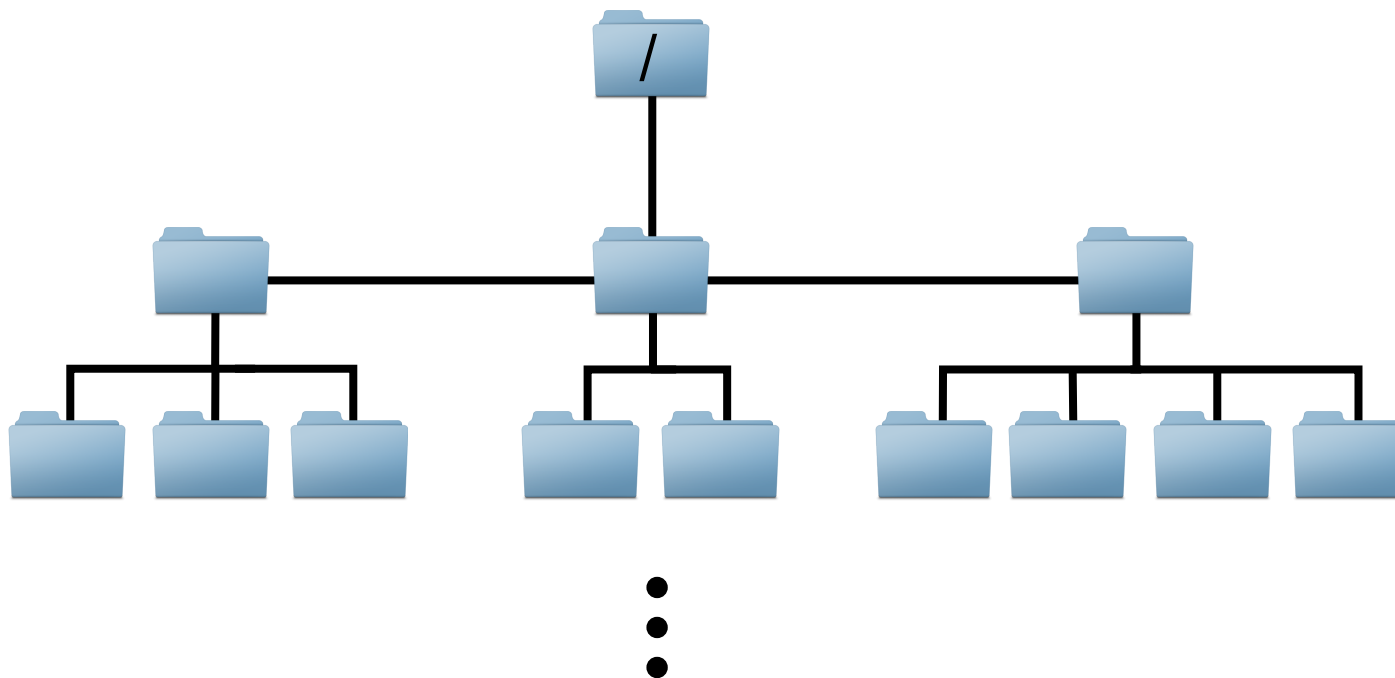
Every file/directory has a specific address, or **path**, in the hierarchy



• The path from C to A is ../..

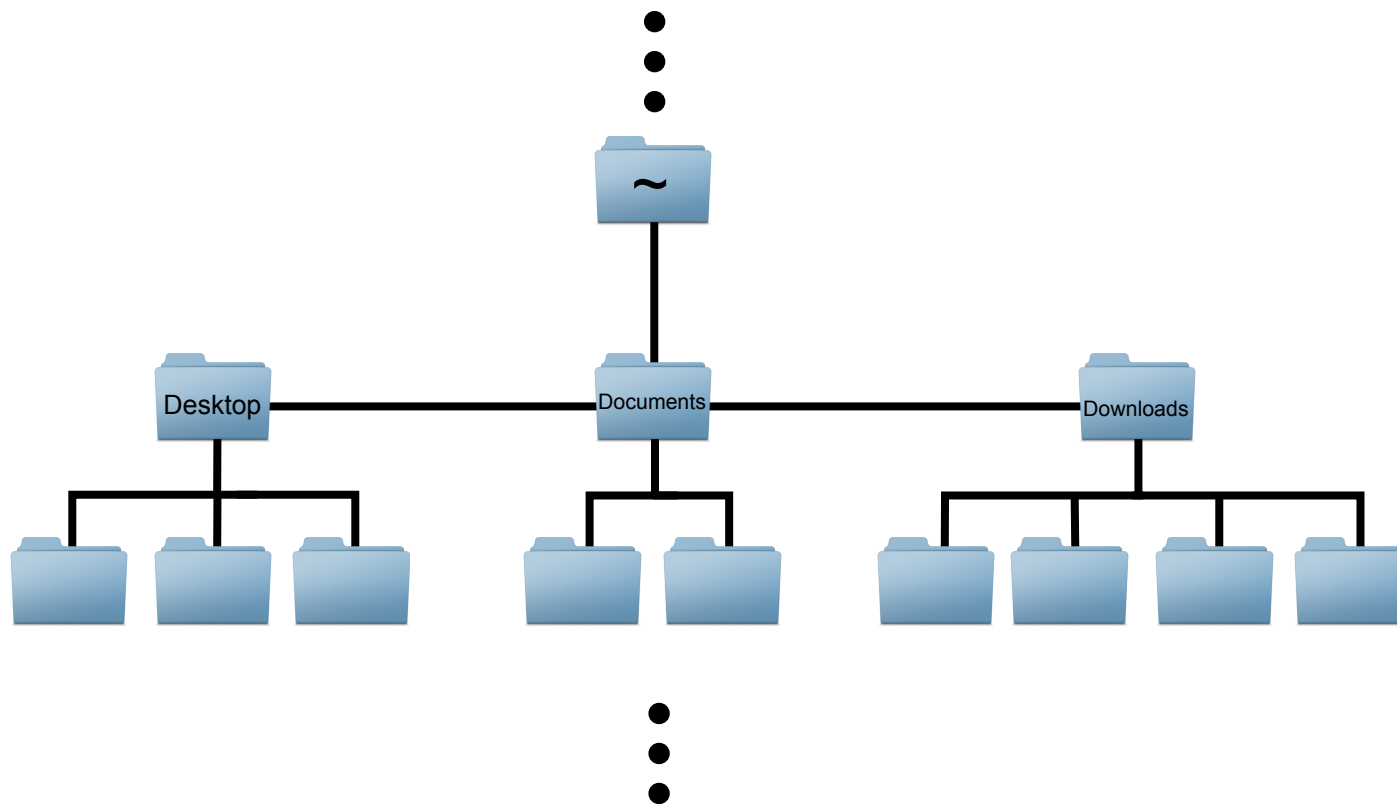
NAVIGATING UNIX SYSTEMS

The *top-level* directory is called *root*, and is denoted with single slash



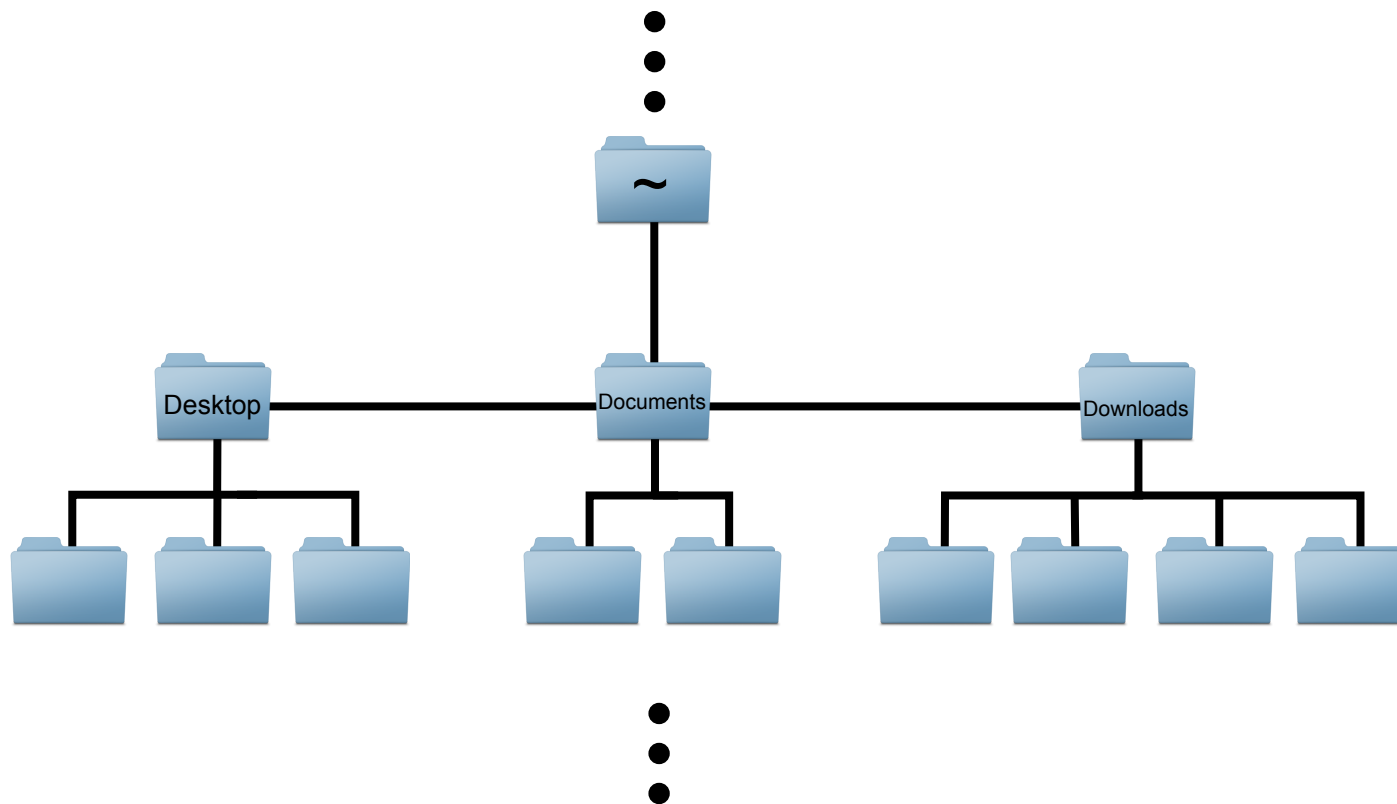
NAVIGATING UNIX SYSTEMS

Your *home directory* is where you live



NAVIGATING UNIX SYSTEMS

Your *home directory* is where you live



The *full path* to my home directory is `/Users/sjspielman/`

NAVIGATING UNIX SYSTEMS

We move around in our file system with the command **cd** (change *directory*)

- Absolute, or full, path is the path *from the root*
- Relative path is the path *from the working directory* (i.e., where you are)

BASIC UNIX COMMANDS

- All UNIX commands are actually *little computer programs*

BASIC UNIX COMMANDS

UNIX Command	What it does
cd	C hange D irectory
pwd	P rint W orking D irectory (gives the full path)
ls	L ist (contents of a directory)
mv	M ove a file or directory (original not retained!)
cp	C opy a file or directory (original <i>is</i> retained)
rm	R emove a file or directory *forever* (NOT A TRASHCAN)
mkdir	M ake a new d irectory
echo	Prints to screen
touch	Create a new (empty) file
man	M anual (shows documentation for a given command)

...And UNIX symbols/shortcuts!

- Typing tab “auto-completes”
- The greater-than sign, >, will re-direct printing to a file (overwrites the file!)
- Two greater-than signs, >>, will also re-direct, but will *append* to the file

ENTER, PYTHON!

- Python is an *interpreted* language
 - We can code either using the interpreter directly or using *scripts* (text files with python code)

ENTER, PYTHON!

- Python is an *interpreted* language
 - We can code either using the interpreter directly or using *scripts* (text files with python code)
- Python is an *object-oriented language*

ENTER, PYTHON!

- Python is an *interpreted* language
 - We can code either using the interpreter directly or using *scripts* (text files with python code)
- Python is an *object-oriented language*

ENTER, PYTHON!

- Python is an *interpreted* language
 - We can code either using the interpreter directly or using *scripts* (text files with python code)
- Python is an *object-oriented language*
 - Objects have *methods* and *attributes*

```
object.method()  
object.attribute
```

PYTHON DATA TYPES

- Each defined variable has a *name*, *value*, and *type*
 - The *type* determines what you can do with the variable

PYTHON DATA TYPES

- Each defined variable has a *name*, *value*, and *type*
 - The *type* determines what you can do with the variable
- We will cover these basic types:
 - Integers
 - Floats
 - Lists
 - Strings
 - Dictionaries (tomorrow)

INTEGERS AND FLOATS

```
# Define some integer variables
```

```
a = 5
```

```
b = -33
```

```
c = 0
```

```
# Define some float variables
```

```
d = 5.67
```

```
e = -33.2
```

```
f = 0.
```

INTEGERS AND FLOATS

```
# Define some integer variables  
a = 5  
b = -33  
c = 0
```

} Integers are *counting numbers*

```
# Define some float variables  
d = 5.67  
e = -33.2  
f = 0.
```

} Floats have *decimals*

INTEGERS AND FLOATS

```
# Define some integer variables
```

```
a = 5
```

```
b = -33
```

```
c = 0
```

```
# Define some float variables
```

```
d = 5.67
```

```
e = -33.2
```

```
f = 0.
```

The **name** of this variable is **f**.

The variable f has a **value** of **0**.

The variable f is a **float data type**.

INTEGERS AND FLOATS

```
# Define some integer variables
```

```
a = 5
```

```
b = -33
```

```
c = 0
```

Comments are denoted with hashtags

```
# Define some float variables
```

```
d = 5.67
```

```
e = -33.2
```

```
f = 0.
```

MATHEMATICAL OPERATIONS

- **Standard symbols**
 - $+$, $-$, $*$, $/$, $\%$

MATHEMATICAL OPERATIONS

- **Standard symbols**

- $+$, $-$, $*$, $/$, $\%$

Define variables and math them

$a = 5$

$b = -33$

$c = a + b$ #c now has a value -28

MATHEMATICAL OPERATIONS

- **Standard symbols**

- $+$, $-$, $*$, $/$, $\%$

Define variables and math them

$a = 5$

$b = -33$

$c = a + b$ #c now has a value -28

What type of variable is c?

MATHEMATICAL OPERATIONS

- **Standard symbols**

- $+$, $-$, $*$, $/$, $\%$

```
# Define variables and math them  
a = 5  
b = -33  
c = a + b  #c now has a value -28
```

```
# Define variables and math them  
a = 5.0  
b = -33  
c = a + b  #c now has a value -28.0
```

MATHEMATICAL OPERATIONS

- **Standard symbols**

- $+$, $-$, $*$, $/$, $\%$

```
# Define variables and math them  
a = 5  
b = -33  
c = a + b  #c now has a value -28
```

```
# Define variables and math them  
a = 5.0  
b = -33  
c = a + b  #c now has a value -28.0
```

What type of variable is c now?

RESULTS FOLLOW THE INPUT TYPES!

```
# Dividing two integers results in an integer
```

```
a = 5
```

```
b = 7
```

```
c = a / b
```

```
# Use print statements to see output
```

```
print c
```

```
0
```

RESULTS FOLLOW THE INPUT TYPES!

Dividing two integers results in an *integer*

```
a = 5
```

```
b = 7
```

```
c = a / b
```

Use print statements to see output

```
print c
```

```
0
```

The solution is to make one of the variables a float

```
a = 5.
```

```
b = 7
```

```
c = a / b
```

```
print c
```

```
0.714285714
```

RESULTS FOLLOW THE INPUT TYPES!

Dividing two integers results in an *integer*

```
a = 5
```

```
b = 7
```

```
c = a / b
```

Use print statements to see output

```
print c
```

```
0
```

The solution is to make one of the variables a float

```
a = 5.
```

```
b = 7
```

```
c = a / b
```

```
print c
```

```
0.714285714
```

```
a = 5
```

```
b = 7
```

```
c = a / float(b)
```

```
print c
```

```
0.714285714
```

MODIFYING THE VALUE IN PLACE

- Mathematical symbols followed by an equals sign will change the variable value *in place*

MODIFYING THE VALUE IN PLACE

- Mathematical symbols followed by an equals sign will change the variable value *in place*
 - $+=$, $-=$, $*=$, $/=$

MODIFYING THE VALUE IN PLACE

- Mathematical symbols followed by an equals sign will change the variable value *in place*

```
# Increment by 5
a = 77
a += 5
print a
      83
```

```
# Multiply by 8
b = 2.5
b *= 8
print b
      20.0
```

PYTHON LISTS

- Lists are defined with brackets: `[]`

PYTHON LISTS

- **Lists are defined with brackets: []**

```
# Define a list  
a = [1, 3, 5, 7, 9]
```

```
# Define another list  
b = [1, 3.1, -5, 7, 9.001]
```

```
# Define another list with *strings* (stay tuned!)  
c = [1, 3.1, -5, 7, 9.001, "woah", "dude"]
```

```
# Define another list..of lists!  
d = [ [1, 2, 3], [11, 22, 33], [7.55, -9] ]
```

PYTHON LISTS

- **Lists are defined with brackets: []**

```
# Define a list  
a = [1, 3, 5, 7, 9]
```

```
# Define another list  
b = [1, 3.1, -5, 7, 9.001]
```

```
# Define another list with *strings* (stay tuned!)  
c = [1, 3.1, -5, 7, 9.001, "woah", "dude"]
```

```
# Define another list..of lists!  
d = [ [1, 2, 3], [11, 22, 33], [7.55, -9] ]
```

What do you notice about the values inside these lists?

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
```

0 1 2 3 4 5 6 ← Indexing starts from 0!

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
```

0 1 2 3 4 5 6 ← Indexing starts from 0!

```
# Index the second entry in d using brackets []  
print d[1]
```

3

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
```

0 1 2 3 4 5 6 ← Indexing starts from 0!

```
# Index the second entry in d using brackets []
print d[1]
```

3

```
# Index a slice of the list with [x:y]
print d[1:4]
```

[3, 5, 7]

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
```

0 1 2 3 4 5 6 ← Indexing starts from 0!

```
# Index the second entry in d using brackets []
print d[1]
```

3

```
# Index a slice of the list with [x:y]
print d[1:4]
```

[3, 5, 7] In [x:y], x is inclusive and y is exclusive

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
```

0 1 2 3 4 5 6 ← Indexing starts from 0!

```
# Index the second entry in d using brackets []
print d[1]
```

3

```
# Index a slice of the list with [x:y]
print d[1:4]
```

[3, 5, 7] In [x:y], x is inclusive and y is exclusive

```
# x and y defaults are 0 and "last index"
```

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
    0  1  2  3  4  5  6    ← Indexing starts from 0!
```

```
# Index the second entry in d using brackets []
print d[1]
      3
```

```
# Index a slice of the list with [x:y]
print d[1:4]
      [3, 5, 7]    In [x:y], x is inclusive and y is exclusive
```

```
# x and y defaults are 0 and "last index"
print d[3:] # assumes go through end of list
      [7, 9, 11, 13]
```

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
    0  1  2  3  4  5  6    ← Indexing starts from 0!
```

```
# Index the second entry in d using brackets []
print d[1]
      3
```

```
# Index a slice of the list with [x:y]
print d[1:4]
      [3, 5, 7]    In [x:y], x is inclusive and y is exclusive
```

```
# x and y defaults are 0 and "last index"
print d[3:] # assumes go through end of list
      [7, 9, 11, 13]
print d[:5] # assumes start at beginning of list
      [1, 3, 5, 7, 9]
```

INDEXING IN PYTHON

```
d = [1, 3, 5, 7, 9, 11, 13]
```

0	1	2	3	4	5	6	← Indexing starts from 0!
-7	-6	-5	-4	-3	-2	-1	← Negative indexing

```
# Index the last entry in d using brackets []  
print d[-1]  
13
```

COMMON LIST METHODS

```
# Define a list  
a = [1, 3, 5, 7, 9, 11, 13]
```

COMMON LIST METHODS

```
# Define a list
a = [1, 3, 5, 7, 9, 11, 13]

# The .append() method adds a value to the end of the list
a.append(15)
print a
    [1, 3, 5, 7, 9, 11, 13, 15]
```

COMMON LIST METHODS

```
# Define a list  
a = [1, 3, 5, 7, 9, 11, 13]
```

```
# The .append() method adds a value to the end of the list  
a.append(15)  
print a  
    [1, 3, 5, 7, 9, 11, 13, 15]
```

```
# The .index() method extracts the index of a given value  
print a.index(3)  
    1
```


COMMON LIST METHODS

```
# Define a list  
a = [1, 3, 5, 7, 9, 11, 13]
```

```
# The .append() method adds a value to the end of the list  
a.append(15)  
print a  
    [1, 3, 5, 7, 9, 11, 13, 15]
```

```
# The .index() method extracts the index of a given value  
print a.index(3)  
    1
```

```
# The .pop() method removes a certain index from the list  
print a.pop(1)  
    [1, 5, 7, 9, 11, 13]
```

COMMON LIST METHODS

```
# Define a list
a = [1, 3, 5, 7, 9, 11, 13]
```

```
# The .append() method adds a value to the end of the list
a.append(15)
print a
    [1, 3, 5, 7, 9, 11, 13, 15]
```

```
# The .index() method extracts the index of a given value
print a.index(3)
    1
```

```
# The .pop() method removes a certain index from the list
print a.pop(1)
    [1, 5, 7, 9, 11, 13]
```

```
print a.pop() # Default behavior removes last index
    [1, 3, 5, 7, 9, 11]
```

PYTHON STRINGS

- Strings are defined with quotes: " " or ' '

PYTHON STRINGS

- **Strings are defined with quotes: " " or ' '**

```
# Define some example strings  
a = "s"  
b = "python"  
c = "I love python!"  
d = "55"
```

PYTHON STRINGS

- **Strings are defined with quotes: " " or ' '**

```
# Define some example strings
```

```
a = "s"
```

```
b = "python"
```

```
c = "I love python!"
```

```
d = "55"
```

Why is d a string and not an integer?

PYTHON STRINGS

- **Strings are defined with quotes: " " or ' '**

```
# Define some example strings  
a = "s"  
b = "python"  
c = "I love python!"  
d = "55"
```

- **We can index strings just like lists**

PYTHON STRINGS

- **Strings are defined with quotes: " " or ' '**

```
# Define some example strings
a = "s"
b = "python"
c = "I love python!"
d = "55"
```

- **We can index strings just like lists**

```
print b[3] # the printed value is also a string
h

print c[:6]
I love
```

COMMON STRING METHODS

```
# Define a string  
s = "This is an example string."
```


COMMON STRING METHODS

```
# Define a string
s = "This is an example string."

# The .upper() method makes the string uppercase
print s.upper()
THIS IS AN EXAMPLE STRING.
```

COMMON STRING METHODS

```
# Define a string  
s = "This is an example string."
```

```
# The .upper() method makes the string uppercase  
print s.upper()  
THIS IS AN EXAMPLE STRING.
```

```
# The .lower() method makes the string lowercase  
print s.lower()  
this is an example string.
```

COMMON STRING METHODS

```
# Define a string  
s = "This is an example string."
```

```
# The .upper() method makes the string uppercase  
print s.upper()  
THIS IS AN EXAMPLE STRING.
```

```
# The .lower() method makes the string lowercase  
print s.lower()  
this is an example string.
```

```
# The .count() method counts occurrences of a given character  
print s.count("i")  
3
```

COMMON STRING METHODS

```
# Define a string  
s = "This is an example string."
```

```
# The .upper() method makes the string uppercase  
print s.upper()  
THIS IS AN EXAMPLE STRING.
```

```
# The .lower() method makes the string lowercase  
print s.lower()  
this is an example string.
```

```
# The .count() method counts occurrences of a given character  
print s.count("i")  
3
```

```
# The .split() method splits on a character, returns a list  
print s.split("i")  
["Th", "s ", "s an example str", "ng."]
```

STRINGS VS LISTS

- **Strings cannot be modified in place**
 - You must *re-define* the string to change it.
- **Lists can be modified in place**

STRINGS ARE IMMUTABLE

```
# Strings *will not* change when methods are called  
s = "example"  
s.upper()  
print s  
    example
```

STRINGS ARE IMMUTABLE

```
# Strings *will not* change when methods are called
s = "example"
s.upper()
print s
    example
```

```
# But if we re-define s when calling .upper()...
s = s.upper()
print s
    EXAMPLE
```

STRINGS ARE IMMUTABLE

```
# Strings *will not* change when methods are called
s = "example"
s.upper()
print s
    example
```

```
# But if we re-define s when calling .upper()...
s = s.upper()
print s
    EXAMPLE
```

```
# We'll get *an error* if we try too much
s[2] = "A"
```

Traceback (most recent call last):
 TypeError: 'str' object does not support item assignment

LISTS ARE MUTABLE

```
# Lists *will* change when methods are called
f = [1, 2, 3]
f.append(4)
print f
    [1, 2, 3, 4]
```

LISTS ARE MUTABLE

```
# Lists *will* change when methods are called
f = [1, 2, 3]
f.append(4)
print f
    [1, 2, 3, 4]
```

```
# We can use indexing to re-write items in lists
f[2] = 77.8
print f
    [1, 2, 77.8, 4]
```

A NOTE ON METHODS

- **Methods are functions *specific to a certain object***
- **Functions are/can be much more general:**

```
s = "python"
print len(s) # number of characters
6
```

```
d = ["p", "y", "t", "h", "o", "n"]
print len(d) # number of items
6
```

A NOTE ON PRINT STATEMENTS

- To print **more than one thing**, join items together with **+**, *as strings*

A NOTE ON PRINT STATEMENTS

```
pi = 3.1415
```

```
print pi  
3.1415
```

```
# To print a string and a float, re-cast the float to a string  
print "The value of pi is" + str(pi)  
The value of pi is 3.1415
```

A NOTE ON PRINT STATEMENTS

```
pi = 3.1415
```

```
print pi  
3.1415
```

```
# To print a string and a float, re-cast the float to a string  
print "The value of pi is" + str(pi)  
The value of pi is 3.1415
```

```
# Print statements that won't work:  
print "The value of pi is" str(pi)  
print "The value of pi is" + pi  
print "The value of pi is" pi  
print "The value of pi is pi"
```

A NOTE ON PRINT STATEMENTS

```
pi = 3.1415
```

```
print pi  
3.1415
```

```
# To print a string and a float, re-cast the float to a string  
print "The value of pi is" + str(pi)  
The value of pi is 3.1415
```

```
# Print statements that won't work:  
print "The value of pi is" str(pi) # missing a +  
print "The value of pi is" + pi    # can't join float and str  
print "The value of pi is" pi      # lotsa problems  
print "The value of pi is pi"      # ok, this works, kinda!
```

A NOTE ON PRINT STATEMENTS

With + and re-casting, we can print anything!

```
myname = "Stephanie"  
myage = 26  
myhometown = "Naples, Florida"
```

```
print "The Intro to Python instructor is named " + myname +  
". She is " + str(myage) + " years old, and she is from " +  
myhometown + "."
```

The Intro to Python instructor is named Stephanie. She is 26 years old, and she is from Naples, Florida.

EXERCISE BREAK