

Python: Introduction and Basics

David Green
UMD NASA/GSFC

Objectives

- Introduce you to the Python language
- Get you writing Python code.
- Convince you its utility in your research life
- Instill good coding and curation practices
- We try not to proselytize, but sometimes it's too hard to resist

Organization

- 4 lectures
 - Basics part 1 and 2, numpy/scipy, working with real data
- Breakout coding sessions after each lecture
- Blood, sweat, tears → a more productive you!

What is



pythonTM

?

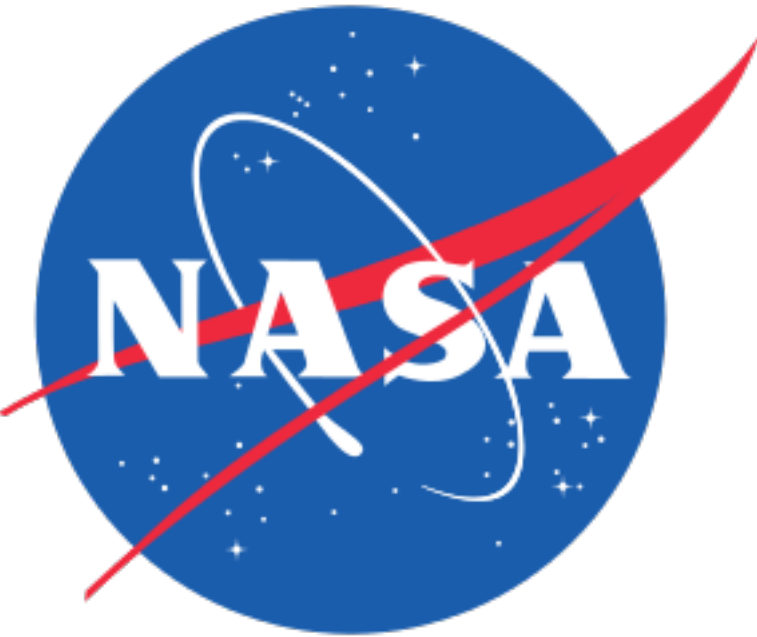
What is Python?

<i>interpreted</i>	no need for a compiling stage
<i>object-oriented</i>	programming paradigm that uses objects (complex data structures with methods)
<i>high level</i>	abstraction from the way machine interprets & executes
<i>dynamic semantics</i>	can change meaning on-the-fly
<i>built in</i>	core language (not external)
<i>data structures</i>	ways of storing/manipulating data
<i>script/glue</i>	programs that control other programs
<i>typing</i>	the sort of variable (int, string)
<i>syntax</i>	grammar which defines the language
<i>library</i>	reusable collection of code
<i>binary</i>	a file that you can run/execute

Why Python?

- **Free**, high portable (Linux, OSX, Windows, ect...)
- **Interactive** interpreter provided
- Extremely readable syntax (“**executable pseudo-code**”)
- **Simple**: non-professional programs can use it effectively
 - Great documentation
 - Total abstraction of memory management
- Rich build-in types: lists, sets, dictionaries (hash tables), strings, ect...
- Standard libraries for IDL/MATLAB-like arrays (NumPy)
- Easy to wrap existing C, C++, and FORTRAN codes

Who uses Python?



digg

IBM

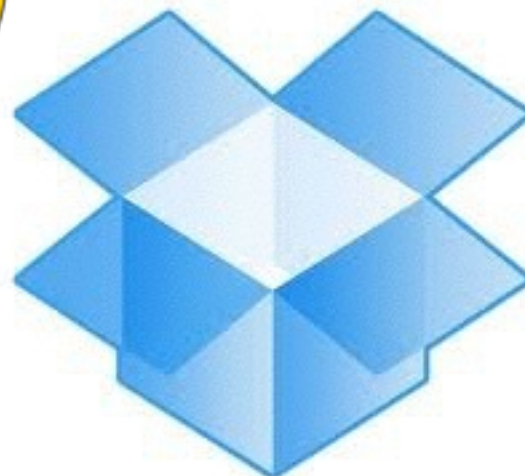
Disney



Google



yelp



Dropbox

Eventbrite



INDUSTRIAL
LIGHT & MAGIC

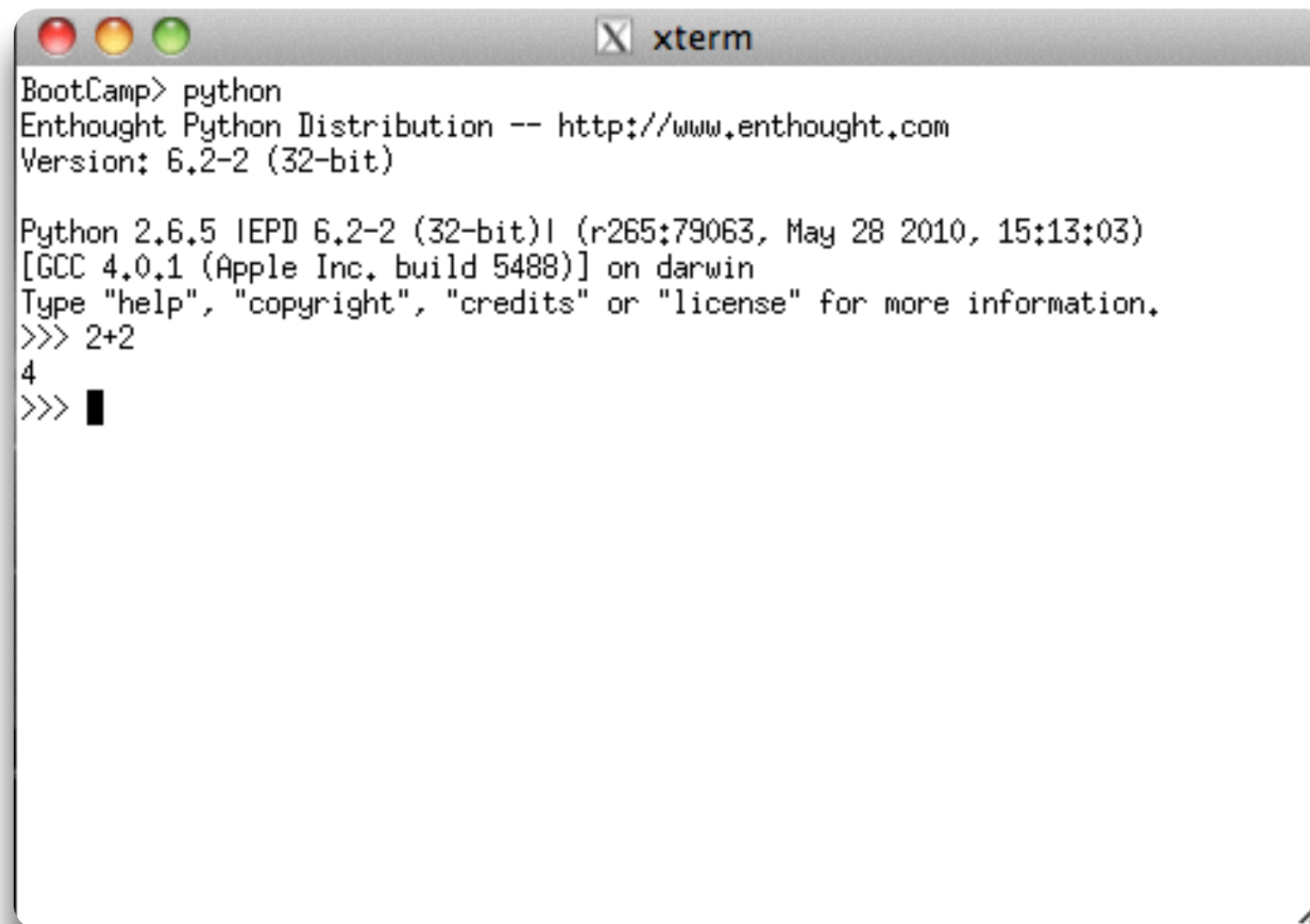


You Tube

USA
United Space Alliance

Ask Questions!!!

Firing up the interpreter

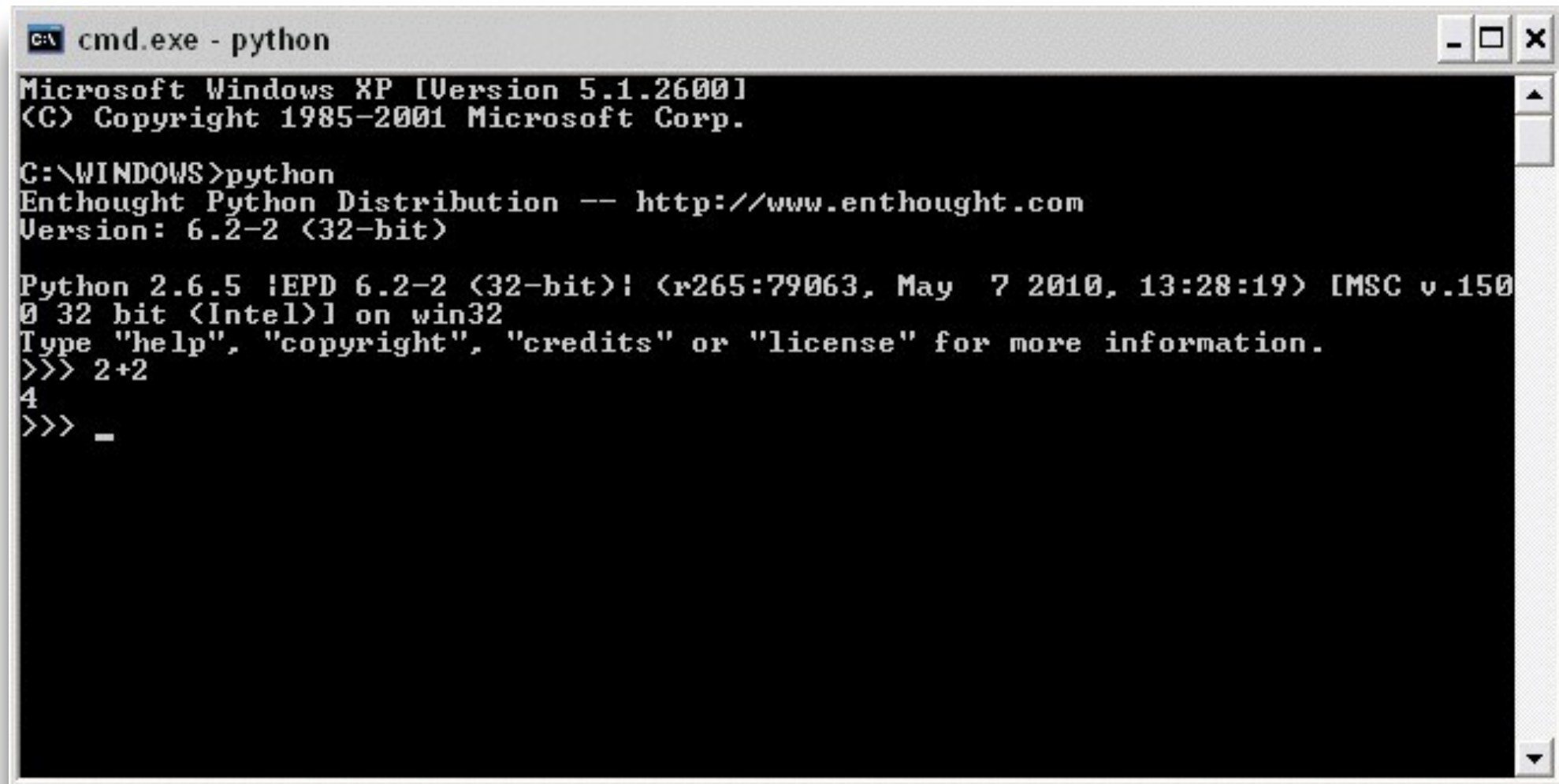


```
BootCamp> python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.2-2 (32-bit)

Python 2.6.5 IEPD 6.2-2 (32-bit) | (r265:79063, May 28 2010, 15:13:03)
[GCC 4.0.1 (Apple Inc. build 5488)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> █
```

Linux/UNIX/Mac OS X (X11/Xterm)

Firing up the interpreter



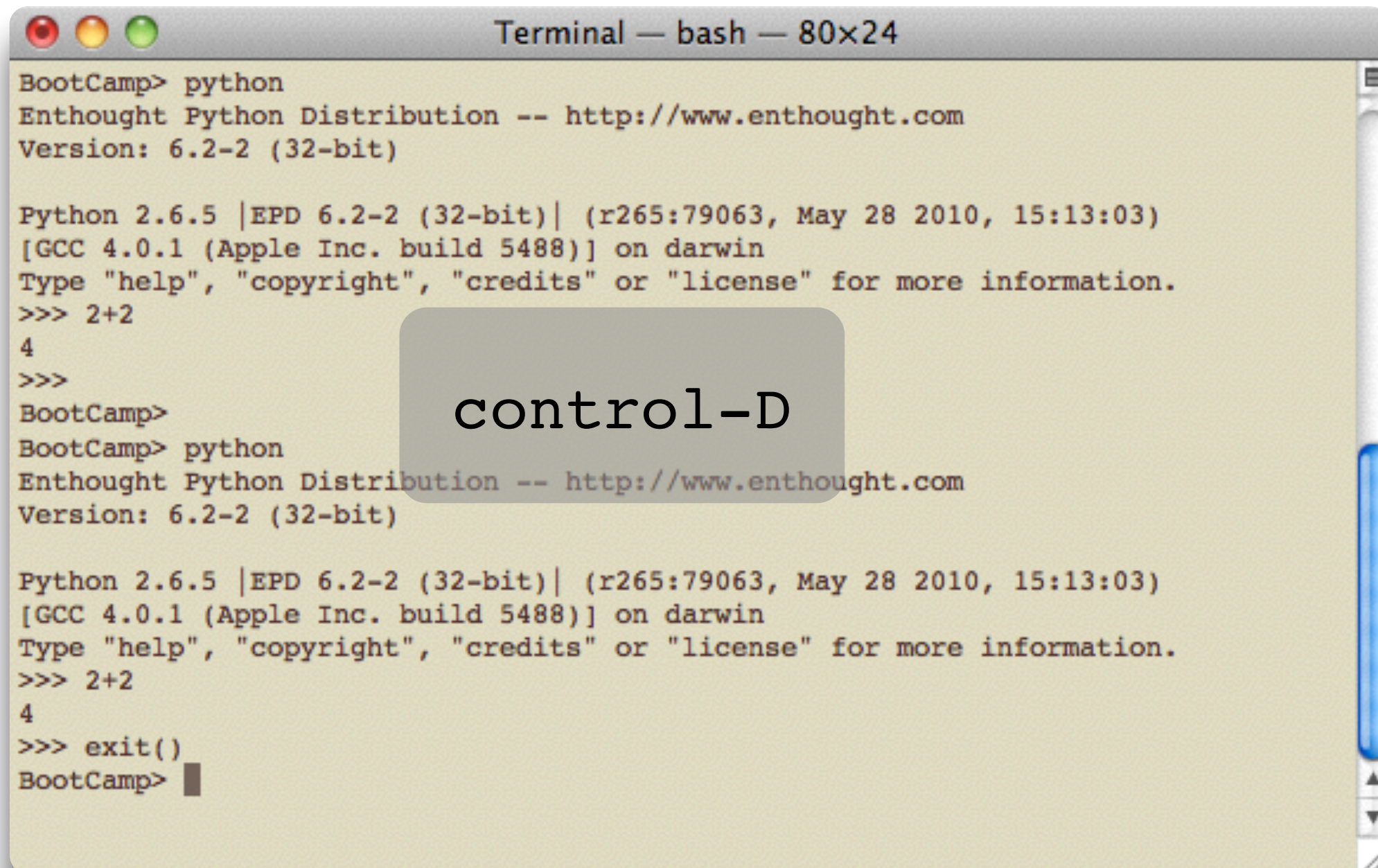
```
cmd.exe - python
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS>python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.2-2 (32-bit)

Python 2.6.5 !EPD 6.2-2 (32-bit)! (r265:79063, May  7 2010, 13:28:19) [MSC v.150
0 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> _
```

Windows

Firing up the interpreter



A screenshot of a macOS Terminal window titled "Terminal — bash — 80x24". The window shows the execution of the Python interpreter. The prompt is "BootCamp>". The user enters "python", which starts the "Enthought Python Distribution" (version 6.2-2, 32-bit). It displays version information: "Python 2.6.5 |EPD 6.2-2 (32-bit)| (r265:79063, May 28 2010, 15:13:03) [GCC 4.0.1 (Apple Inc. build 5488)] on darwin". It prompts the user to type "help", "copyright", "credits", or "license" for more information. The user enters ">>> 2+2", and the interpreter returns "4". The user enters ">>>" and the prompt returns to "BootCamp>". The user enters "python" again, and the same version information is displayed. The user enters ">>> 2+2", and the interpreter returns "4". The user enters ">>> exit()", and the prompt returns to "BootCamp>". A semi-transparent grey box with the text "control-D" is overlaid on the terminal output.

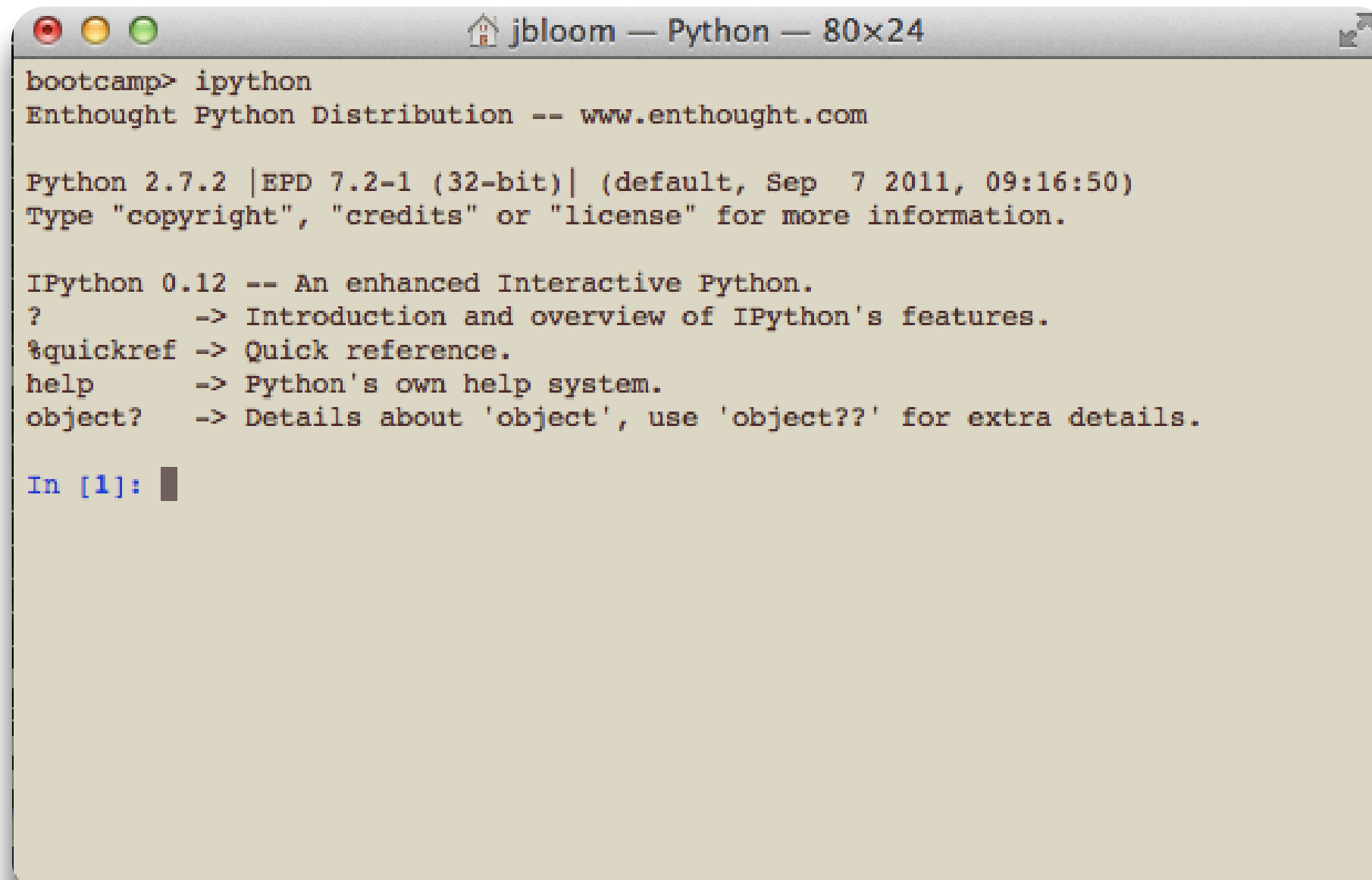
```
BootCamp> python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.2-2 (32-bit)

Python 2.6.5 |EPD 6.2-2 (32-bit)| (r265:79063, May 28 2010, 15:13:03)
[GCC 4.0.1 (Apple Inc. build 5488)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>>
BootCamp>
BootCamp> python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.2-2 (32-bit)

Python 2.6.5 |EPD 6.2-2 (32-bit)| (r265:79063, May 28 2010, 15:13:03)
[GCC 4.0.1 (Apple Inc. build 5488)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> exit()
BootCamp>
```

to exit: either `control-D` or `exit()`

Firing up the interpreter



```
jbloom — Python — 80x24
bootcamp> ipython
Enthought Python Distribution -- www.enthought.com

Python 2.7.2 |EPD 7.2-1 (32-bit)| (default, Sep  7 2011, 09:16:50)
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]:
```

iPython

BASIC TRAINING



Outline

- Hello World!
- Calculator/basic math
- Strings
- Variables
- Basic control statements
 - Indentation!

Hello, World.



C++

file: hello.cpp

```
#include <iostream>
int main()
{
    std::cout << "Hello World!" << std::endl;
}
```

```
BootCamp> g++ -o hello hello.cpp
BootCamp> ./hello
Hello World!
BootCamp>
```

FORTRAN

file: hello.f

```
PROGRAM HELLO
WRITE (*,100)
STOP
100 FORMAT ( ' Hello World! ' /)
END
```

```
BootCamp> g77 -o hello hello.f
BootCamp> ./hello
Hello World!
BootCamp>
```

Java

file: hello.java

```
class HelloWorld {
    static public void main( String args[] ) {
        System.out.println( "Hello World!" );
    }
}
```

```
BootCamp> javac hello.java
BootCamp> java HelloWorld
Hello World!
BootCamp>
```

Examples of
compiled languages

Scripted

file: hello.py

```
print "Hello World!"
```

```
BootCamp> python hello.py  
Hello World!  
BootCamp>
```

Interactive

```
BootCamp> python  
>>> print "Hello World!"  
Hello World!  
>>>
```

2 Points

1. Python provides both an interactive way to develop code and a way to execute scripts
2. What you do interactively is basically the same things you (can) do in your scripts

Calculator

```
>>> print 2 + 2
4
>>> 2 + 2
4
>>> print 2.1 + 2
4.1
>>> 2.1 + 2 == 4.0999999999999996
True
```

- Python has int and floats (but not natively doubles)
- Python stores floats as their byte representation so it's limited by the same 16-bit issues as most other languages
- In doing calculations, unless you specify otherwise, Python will store the results in the smallest-byte representation

Indentation

- Indentation matters!!!
- When you mess up, python is gentle
- # starts a comment (until the end of the line)

```
>>> 2 + 2
```

```
4
```

```
>>> 2 + 2
```

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

```
2 + 2
```

```
^
```

```
IndentationError: unexpected indent
```

```
>>> 2 # this is a comment and is not printed
```

```
2
```

```
>>> # this is also a comment
```

```
>>>
```



handy error message!

Calculator

```
>>> (3.0*10.0 - 25.0)/5.0
1.0
>>> print 3.085e18*1e6 # this is a Megaparsec in units of cm!
3.085e+24
>>> t = 1.0 # declare a variable t (time)
>>> accel = 9.8 # acceleration in units of m/s^2
>>> # distance travelled in time t seconds is 1/2 a*t**2
>>> dist = 0.5*accel*t*t
>>> print dist # this is the distance in meters
4.9
>>> dist1 = accel*(t**2)/2
>>> print dist1
4.9
>>> dist2 = 0.5*accel*pow(t,2)
>>> print dist2
4.9
```

- **Variables** are assigned on the fly
- Multiplication, division, exponents work as you expect

Calculator

Relationships

```
>>> # from before dist1 = 4.9 and dist = 4.9
```

```
>>> dist1 == dist
```

```
True
```

```
>>> dist < 10
```

```
True
```

```
>>> dist <= 4.9
```

```
True
```

```
>>> dist < (10 + 2j)
```

```
-----  
TypeError
```

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

```
/Users/damgreen/<ipython console> in <module>()
```

```
TypeError: no ordering relation is defined for complex numbers
```

```
>>> dist < -2.0
```

```
False
```

```
>>> dist != 3.1415
```

```
True
```

More on Variables and Types

None, numbers, and truth

```
>>> 0 == False
True
>>> not False
True
>>> 0.0 == False
True
>>> not (10.0 - 10.0)
True
>>> not -1
False
>>> not 3.1415
False
>>> x = None      # None is something special. Not true or false
>>> None == False
False
>>> None == True
False
>>> False or True
True
>>> False and True
False
```

More on Variables and Types

```
>>> print type(1)
<type 'int'>
>>> x = 2 ; type(x)
<type 'int'>
>>> type(2) == type(1)
True
>>> print type(True)
<type 'bool'>
>>> print type(type(1))
<type 'type'>
>>> print type(pow)
<type 'builtin_function_or_method'>
```

We can test whether something is a certain type with `isinstance()`

```
>>> isinstance(1,int)
True
>>> isinstance("spam",str)
True
>>> isinstance(1.212,int)
False
```

builtin-types: `int`, `bool`, `str`, `float`, `complex`, `long`,

Strings

- Strings are a sequence of characters
 - They can be indexed and sliced up if they were an array
 - you can glue strings together with + signs
- Strings are **immutable** (unlike in C), so you cannot change a string in place (not as bad as it sounds)
- Strings can be formatted and compared

Strings

```
>>> x = "spam" ; print type(x)
<type "str">
>>> print "hello!\n...my sire."
hello!
...my sire.
>>> "hello!\n...my sire."
'hello!\n...my sire.'
>>> "wah?!" == 'wah?!'
True
>>> print "'wah?!' said the student"
'wah?!' said the student
>>> print "\"wah?!\" said the student"
"wah?!" said the student
```

- backslashes (\) start special (escape) characters:
 - \n = newline (\r = return)
 - \t = tab
 - \a = bell
- String literals are defined with double quotes or quotes.
- The outermost quote type cannot be used inside the strings (unless it's escaped with a backslash)

Strings

```
>>> s = "spam" ; e = "eggs"
>>> print s + e
spameggs
>>> print s + " and " + e
spam and eggs
>>> print "green " + e + " and\n " + s
green eggs and
spam
>>> print s*3 + e
spamspamspameggs
>>> print "*" * 50
*****
>>> print "spam" is "good" ; print "spam" is "spam"
False
True
>>> "spam" < "zoo"
True
>>> "s" < "spam"
True
```

- You can concatenate strings with + sign
- You can do multiple concatenations with the * sign
- Strings can be compared

Strings

```
>>> print 'I want' + 3 + ' eggs and no ' + s
```

```
-----  
TypeError                                Traceback (most recent call last)
```

```
/Users/damgreen/<ipython console> in <module>()
```

```
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> print 'I want ' + str(3) + ' eggs and no ' + s
```

```
I want 3 eggs and no spam
```

```
>>> pi = 3.14159
```

```
>>> print 'I want ' + str(pi) + ' eggs and no ' + s
```

```
I want 3.14159 eggs and no spam
```

```
>>> print str(True) + ":" + ' I want ' + str(pi) + ' eggs and no ' + s
```

```
True: I want 3.14159 eggs and no spam
```

You must concatenate only strings, coercing (“casting”) others
variables to `str`

Strings

Getting input from the user: always a string response

```
>>> faren = raw_input("Enter the temperature (in Fahrenheit): ")
Enter the temperature (in Fahrenheit): 71
>>> cent = (5.0/9.0)*(faren - 32.0)
...
TypeError: unsupported operand type(s) for -: 'str' and 'float'
>>> faren = float(faren)
>>> cent = (5.0/9.0)*(faren - 32.0) ; print cent
21.6666666667
>>> faren = float(raw_input("Enter the temperature (in Fahrenheit): "))
Enter the temperature (in Fahrenheit): 71
>>> print (5.0/9.0)*(faren - 32.0)
21.6666666667
>>> faren = float(raw_input("Enter the temperature (in Fahrenheit): "))
Enter the temperature (in Fahrenheit): meh!
...
ValueError: invalid literal for float(): meh!
```

Strings

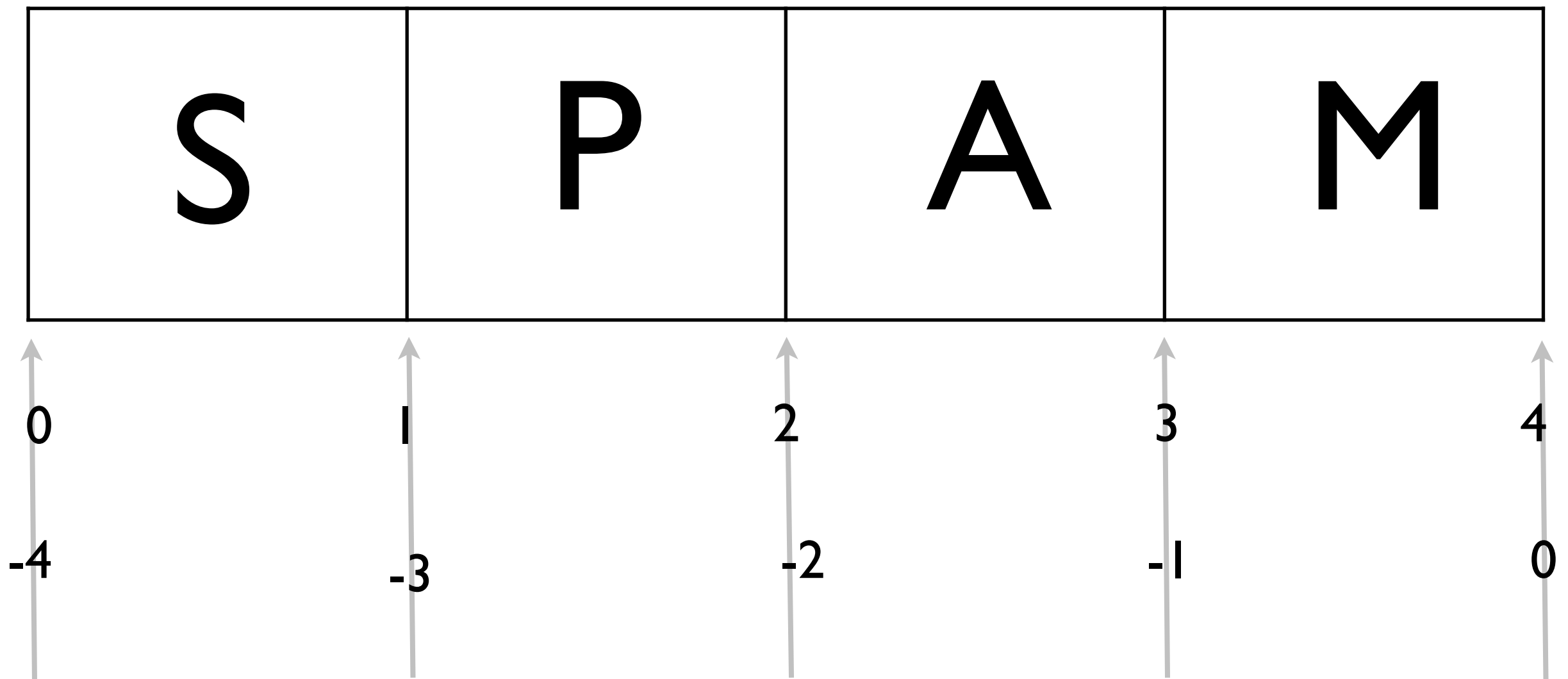
We can think of strings as arrays (although, unlike in C you never really need to deal with directly addressing characters locations in memory)

```
>>> s = "spam"
>>> len(s)
4
>>> len("eggs\n")
5
>>> len("")
0
>>> s[0]
's'
>>> s[-1]
'm'
```

- len() gives us the length of an array
- strings are zero indexed
- Can also count backwards

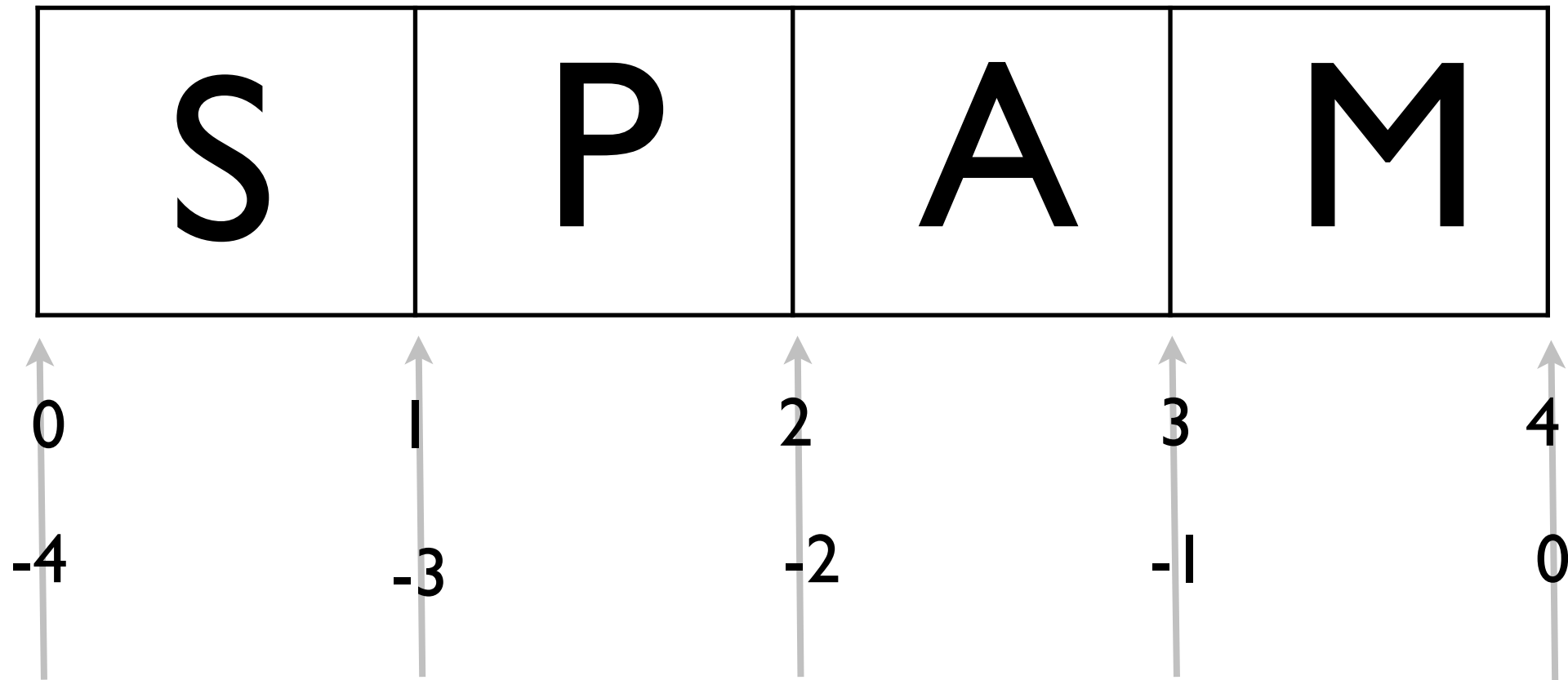
Strings

We can think of strings as arrays (although, unlike in C you never really need to deal with directly addressing characters locations in memory)



Useful for slicing: indices are between the characters

Strings



```
>>> s[0:1] # get every character between 0 and 1
's'
>>> s[1:4] # get every character between 1 and 4
'pam'
>>> s[-2:-1]
'a'
>>> ## slicing [m:n] will return abs(n-m) characters
>>> s[0:100] # if the index is beyond the len(str), you dont segfault!
'spam'
>>> s[1:] # python runs the index to the end
'pam'
>>> s[:2] # python runs the index to the beginning
'sp'
```

$s = s[:n] + s[n:]$ for all n

Strings can do operations on themselves:

`.lower()`, `.upper()`, `.capitalize()`

```
>>> "funKY tOwn".capitalize()
'Funky town'
>>> "funKY tOwn".lower()
'funky town'
```

`.split([sep [,maxsplit]])`

```
>>> "funKY tOwn".split()
['funKY', 'tOwn']
>>> "funKY tOwn".capitalize().split()
['Funky', 'town']
>>> [x.capitalize() for x in "funKY tOwn".split()]
['Funky', 'Town']
>>> "I want to take you to, funKY tOwn".split("u")
['I want to take yo', ' to, f', 'nKY tOwn']
>>> "I want to take you to, funKY tOwn".split("you")
['I want to take ', ' to, funKY tOwn']
```


.strip(), .join(), .replace()

```
>>> csv_string = 'Dog,Cat,Spam,Defenestrate,1, 3.1415 \n\t'
>>> csv_string.strip()
'Dog,Cat,Spam,Defenestrate,1, 3.1415'
>>> clean_list = [x.strip() for x in csv_string.split(",")]
>>> clean_list
['Dog', 'Cat', 'Spam', 'Defenestrate', '1', '3.1415']
```

.join() allows you to glue a list of strings together with a certain string

```
>>> print ",".join(clean_list)
'Dog,Cat,Spam,Defenestrate,1,3.1415'
>>> print "\t".join(clean_list)
Dog  Cat  Spam Defenestrate  1    3.1415
```

.replace() strings in strings

```
>>> csv_string = 'Dog,Cat,Spam,Defenestrate,1, 3.1415 \n\t'
>>> alt_csv = csv_string.strip().replace(' ','')
>>> alt_csv
'Dog,Cat,Spam,Defenestrate,1,3.1415'
>>> print csv_string.strip().replace(' ','').replace(',','\t')
Dog  Cat  Spam Defenestrate  1    3.1415

'Dog,Cat,Spam,Defenestrate,1,3.1415'
```

`.find()`

*incredibly useful searching,
returning the index of the search*

```
>>> s = 'My Funny Valentine'
>>> s.find("y")
1
>>> s.find("y",2)
7
>>> s[s.find("Funny"):]
'Funny Valentine'
>>> s.find("z")
-1
>>> ss = [s,"Argentine","American","Quarentine"]
>>> for thestring in ss:
    if thestring.find("tine") != -1:
        print "'" + str(thestring) + "' contains 'tine'."

'My Funny Valentine' contains 'tine'.
'Argentine' contains 'tine'.
'Quarentine' contains 'tine'.
>>>
```

There are four main types of collections of data: ("Sequence objects")

- Tuples: ordered, immutable list
- List: a mutable array of data
- Sets: unordered collection of unique elements
- Dictionary: keyword/value lookup

The value in each element can be whatever
(type) you want

Tuple

denoted with parentheses

```
>>> t = (12,-1)
>>> print type(t)
<type "tuple">
>>> isinstance(t,tuple)
True
>>> len(t)
2
>>> t = (12,"monty",True,-1.23e6)
>>> t[1]
'monty'
>>> t[-1]
-1.23e6
>>> t[-2:] # get the last two elements, return as a tuple
(True, -1230000.0)
>>> x = (True) ; type(x)
<type "bool">
>>> x = (True,) ; type(x)
<type "tuple">
>>> type(()), len(())
(<type "tuple">, 0)
```

single-element tuples look like `(element,)`

Tuple

cannot change a tuple
but you can create new one with concatenation

```
>>> t[2] = False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t[0:2], False, t[3:]
((12, 'monty'), True, (-1230000.0,))
>>> ## not what we wanted... need to concatenate
>>> t[0:2] + False + t[3:]
TypeError: can only concatenate tuple (not "bool") to tuple
>>> y = t[0:2] + (False,) + t[3:] ; print y
(12, 'monty', False, -1230000.0)
>>> t*2
(12, 'monty', True, -1230000.0, 12, 'monty', True, -1230000.0)
```

List

denoted with a brackets

```
>>> v = [1,2,3] ; print len(v), type(v)
(3,<type "list">
>>> v[0:2]
[1,2]
>>> v = ["eggs","spam",-1,("monty","python"),[-1.2,-3.5]]
>>> len(v)
5
>>> v[0] ="green egg"
>>> v[1] += ",love it."
['green egg', 'spam,love it.', -1, ('monty', 'python'), [-1.2, -3.5]]
>>> v[-1]
[-1.2, -3.5]
>>> v[-1][1] = None ; print v
['green egg', 'spam,love it.', -1, ('monty', 'python'), [-1.2, None]]
>>> v = v[2:] ; print v
[-1, ('monty', 'python'), [-1.2, None]]
>>> # let's make a proto-array out of nested lists
>>> vv = [ [1,2], [3,4] ]
>>> determinant = vv[0][0]*vv[1][1] - vv[0][1]*vv[1][0]
```

lists are changeable

List

lists can be extended & appended

```
>>> v = [1,2,3]
>>> v.append(4)
>>> print v
[1,2,3,4]
```

Lists can be considered **objects**.

Objects are like animals: they know how to do stuff (like eat and sleep), they know how to interact with others (like make children), and they have characteristics (like height, weight).

"Knowing how to do stuff" with itself is called a **method**. In this case "append" is a method which, when invoked, is an action that changes the characteristics (the data vector of the list itself).

List

lists can be extended, appended, and popped

```
>>> v.append([-5])
>>> print v
[1,2,3,4,[-5]]
>>> v = v[:4]
>>> w = ['elderberries', 'eggs']
>>> v + w
[1,2,3,4,'elderberries','eggs']
>>> v.extend(w) ; print v
[1,2,3,4,'elderberries','eggs']
>>> v.pop()
'eggs'
>>> print v
[1,2,3,4,'elderberries']
>>> v.pop(0) ; print v ## pop the first element
1
[2, 3, 4, 'elderberries']
```

- **append ()**: adds a new element
- **extend ()**: concatenates a list/element
- **pop ()**: remove an element

List

lists can be searched, sorted, & counted

```
>>> v = [1,3, 2, 3, 4, 'elderberries']
>>> v.sort() ; print v
[1, 2, 3, 3, 4, 'elderberries']
>>> v.sort(reverse=True) ; print v
['elderberries', 4, 3, 3, 2, 1]
>>> v.index(4)    ## lookup the index of the entry 4
1
>>> v.index(3)    # get the first occurrence of the number 3
2
>>> v.count(3)
2
>>> v.insert(0,"it's full of stars") ; print v
["it's full of stars", 'elderberries', 4, 3, 3, 2, 1]
>>> v.remove(1) ; print v
["it's full of stars", 'elderberries', 4, 3, 3, 2]
```

reverse is a keyword of the `.sort()` method

Exercise for the Breakout

Write a program that given a height, angle, and initial velocity finds distance and time traveled by a projectile.

Extra: Calculate and print using proper significant figures

Extra: Find the distance traversed for every second

Example interaction:

```
Input starting height (m): 10
Input starting angle (degrees): 30
Input starting velocity (m/s): 20
Range: 48.04 m
Time of flight: 2.77 s
Time: 0.0 s, Range : 0.0 m
Time: 1.0 s, Range : 17.32 m
Time: 2.0 s, Range : 34.64 m
```