

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

- 5 lectures
 - Basics, Data structures, Functions and Strings, Advanced misc, Science packages (numpy, pylab, scipy, ect...)
- Breakout coding sessions after each lecture
- Homework (small code projects)
- Blood, sweat, tears → a more productive you!

What is



?

What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

<http://www.python.org/doc/essays/blurb/>

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

Development History

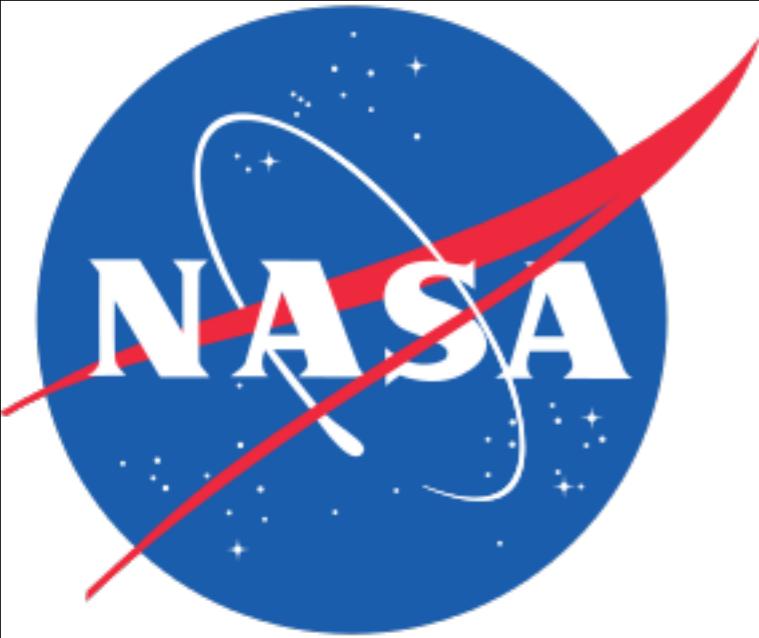
- Open-sourced development from the start (BSD licensed now)
<http://www.opensource.org/licenses/bsd-license.php>
- Relies on large community input (bugs, patches) and 3rd party add-on software
- Version 2.0 (2000), 2.6 (2008), 2.7 (2010).
We're using **2.7.3** in this class
- Version 3.X (2008) is not backward compatible with 1.X & 2.X. But 2.7 code is “easily” migrated to 3.X

Why Python?

- **C, C++, FORTRAN**
 - Pros: great performance, backbone of legacy scientific computing codes
 - Cons: syntax not optimized for causal programming, no interactive facilities, difficult visualization, text processing, etc.
- **Mathematica, Maple, MATLAB, IDL**
 - Pros: Interactive, great visuals, extensive libraries
 - Cons: Costly, proprietary, unpleasant for large-scale programs and non-mathematical task

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

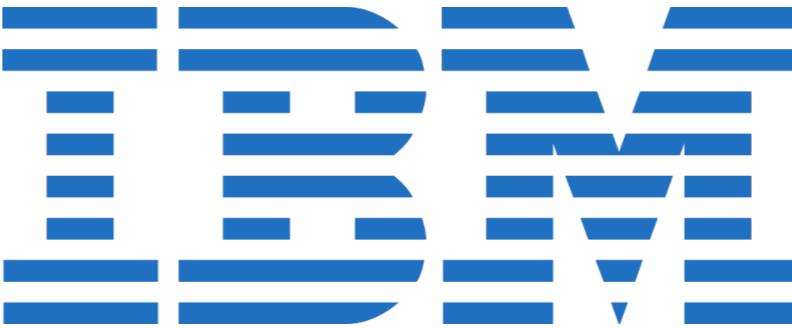
Disney

Google™

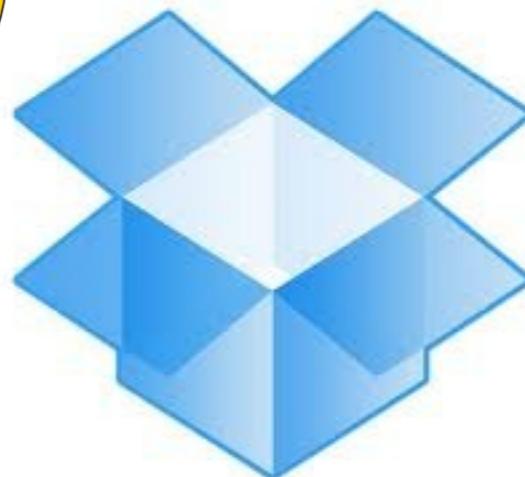
Eventbrite®

You Tube

USA
United Space Alliance



yelp



Dropbox

INDUSTRIAL
LIGHT & MAGIC



Interactive Notebooks

Untitled (Sage)

http://localhost:8000/home/admin/1/ Google

f = u.function(x, y); f

$$(x, y) \mapsto \log\left(\frac{2-x}{y+5}\right)$$

```
x = var("x")
y = x^2
dy = derivative(y, x)
z = integral(sqrt(1 + dy^2), x, 0, 2)
print z
```

$$\frac{\operatorname{arcsinh}(4) + 4 \sqrt{17}}{4}$$

```
I = CDF.0
show(line([zeta(1/2 + k*I/6) for k in xrange(180)],
rgbcolor=(3/4, 1/2, 5/8)))
```

jsMath

IP[y]: Notebook

Untitled0

Save

QuickHelp

Notebook

Actions

[New](#) [Open](#)
[Download](#) [ipynb](#) [Print](#)

Cell

Actions

[Delete](#)
Format [Code](#) [Markdown](#)
Output [Toggle](#) [ClearAll](#)
Insert [Above](#) [Below](#)
Move [Up](#) [Down](#)
Run [Selected](#) [All](#)
Autoindent:

Kernel

Actions

[Interrupt](#) [Restart](#)
Kill kernel upon exit:

Help

Links

[Python](#) [IPython](#)
[NumPy](#) [SciPy](#)
[MPL](#) [SymPy](#)

Shift-Enter : run selected cell

Ctrl-Enter : run selected cell in-place

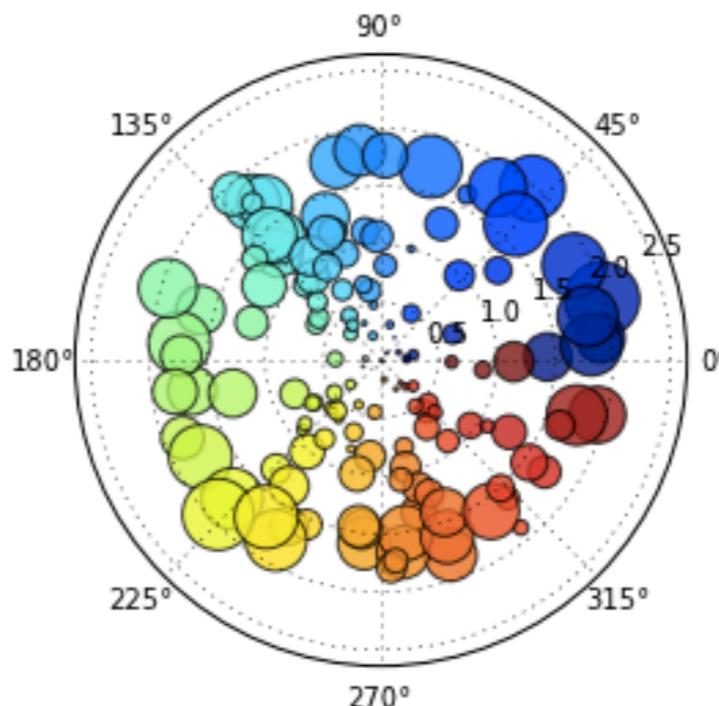
Ctrl-m h : show keyboard shortcuts

Configuration

Tooltip on tab:

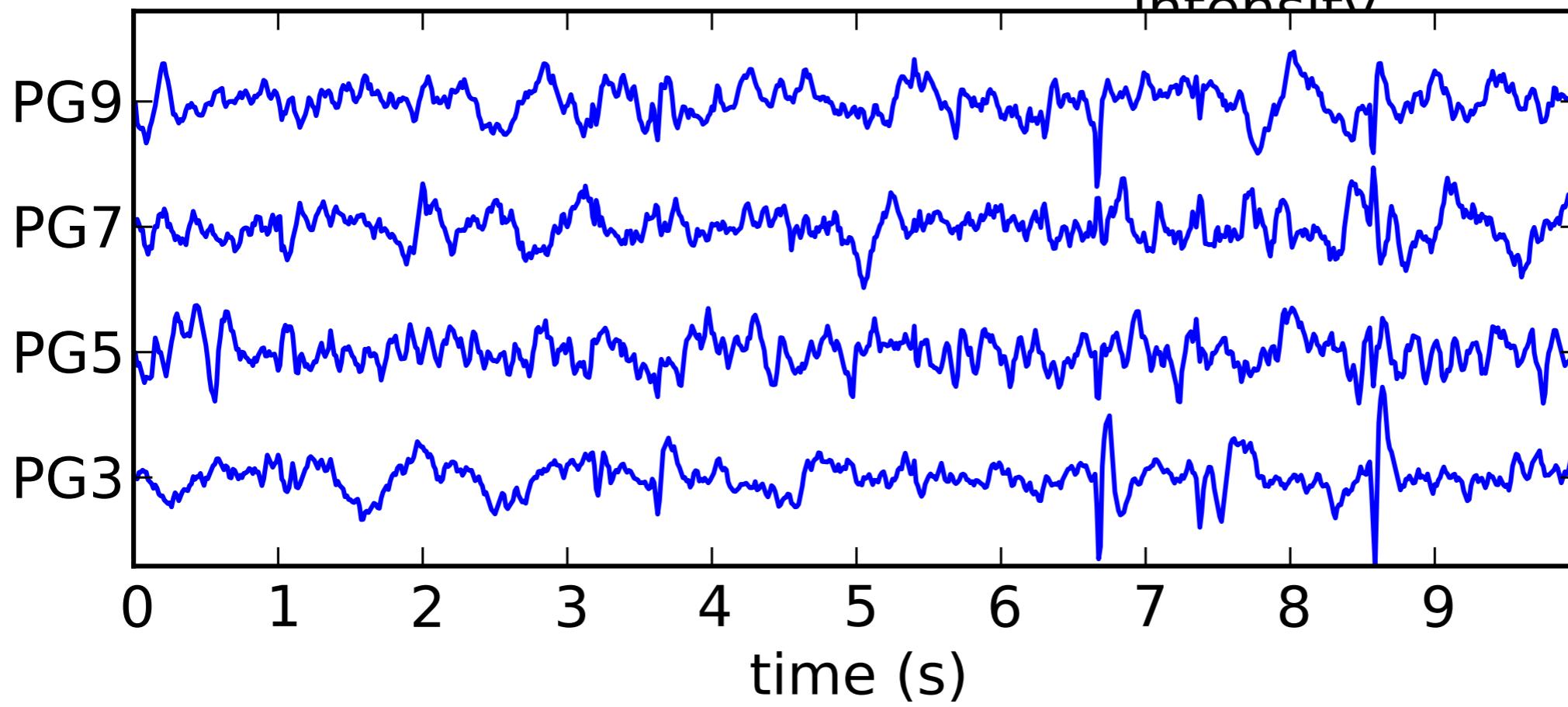
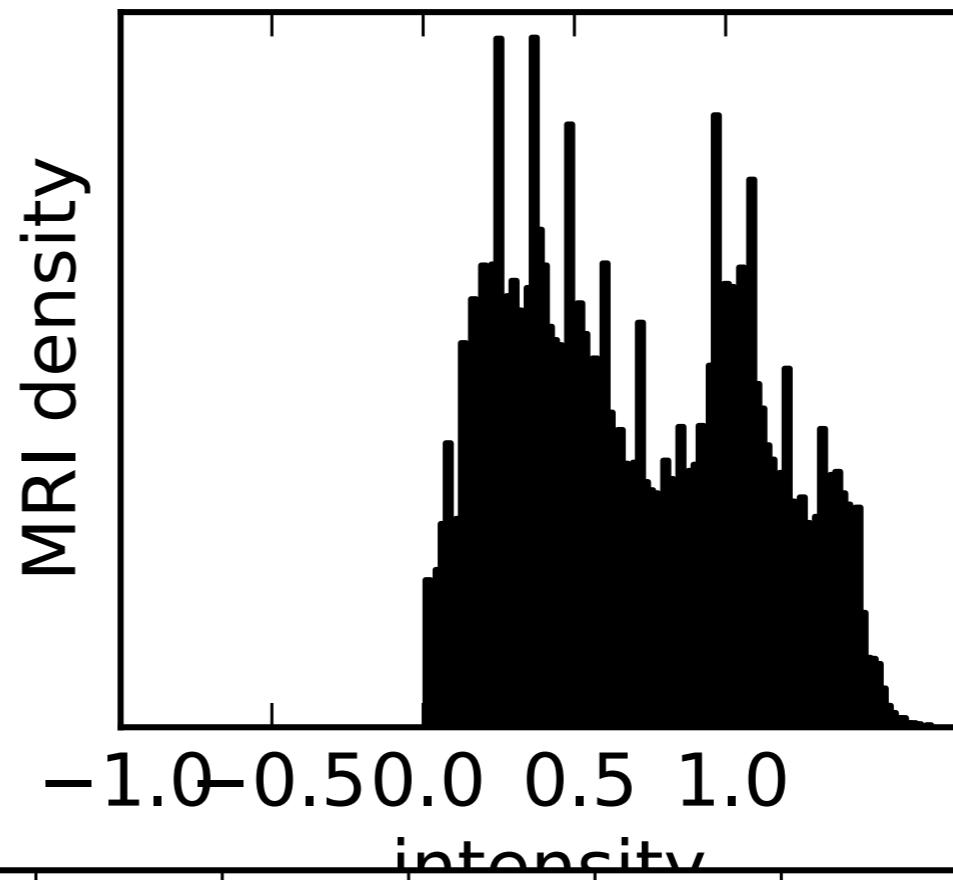
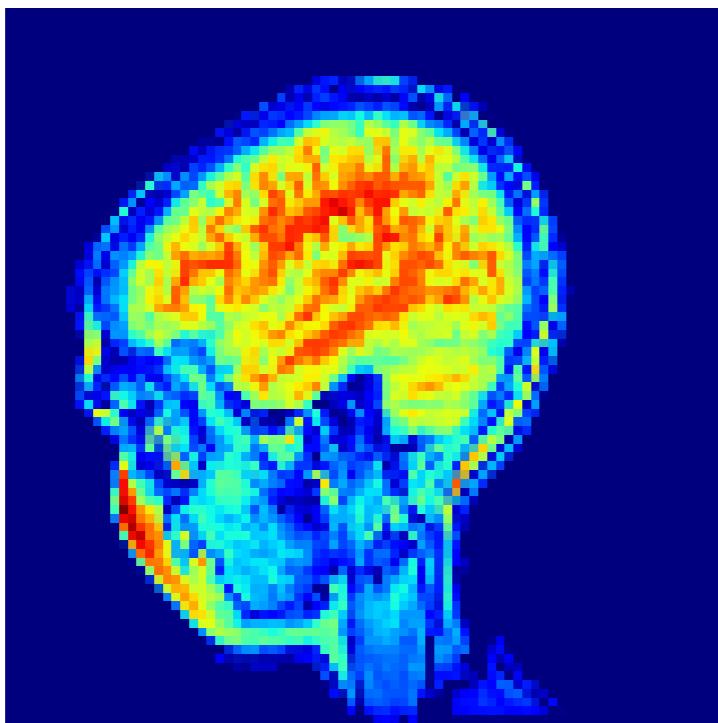
In [1]:

```
num_bootcampers=142
# note: rand not Rnd
r = 2*rand(num_bootcampers)
theta = 2*pi*rand(num_bootcampers)
area = 200*r**2*rand(num_bootcampers)
colors = theta
ax = subplot(111, polar=True)
c = scatter(theta, r, c=colors, s=area)
c.set_alpha(0.75)
```

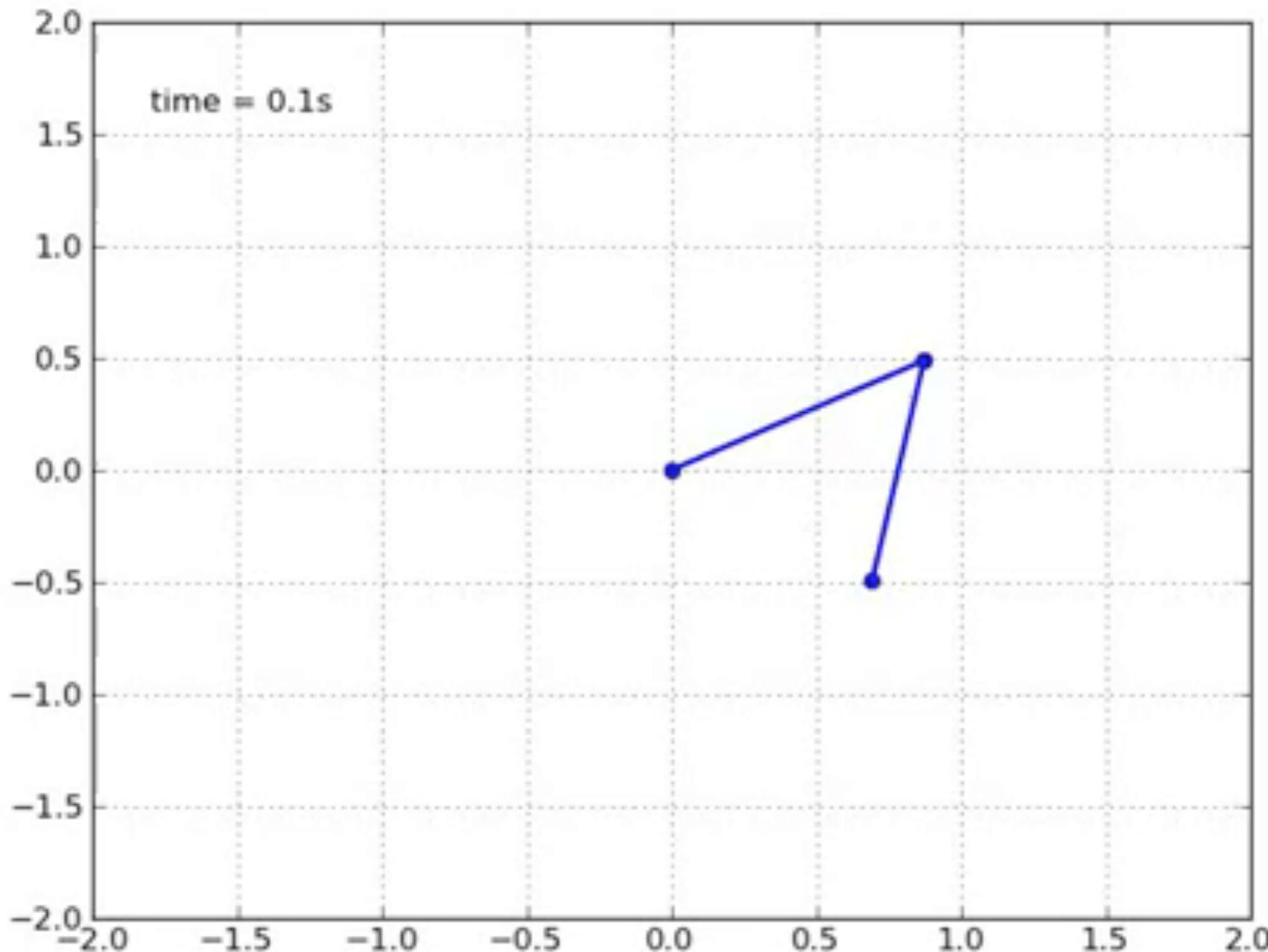


In []:

iPython notebook

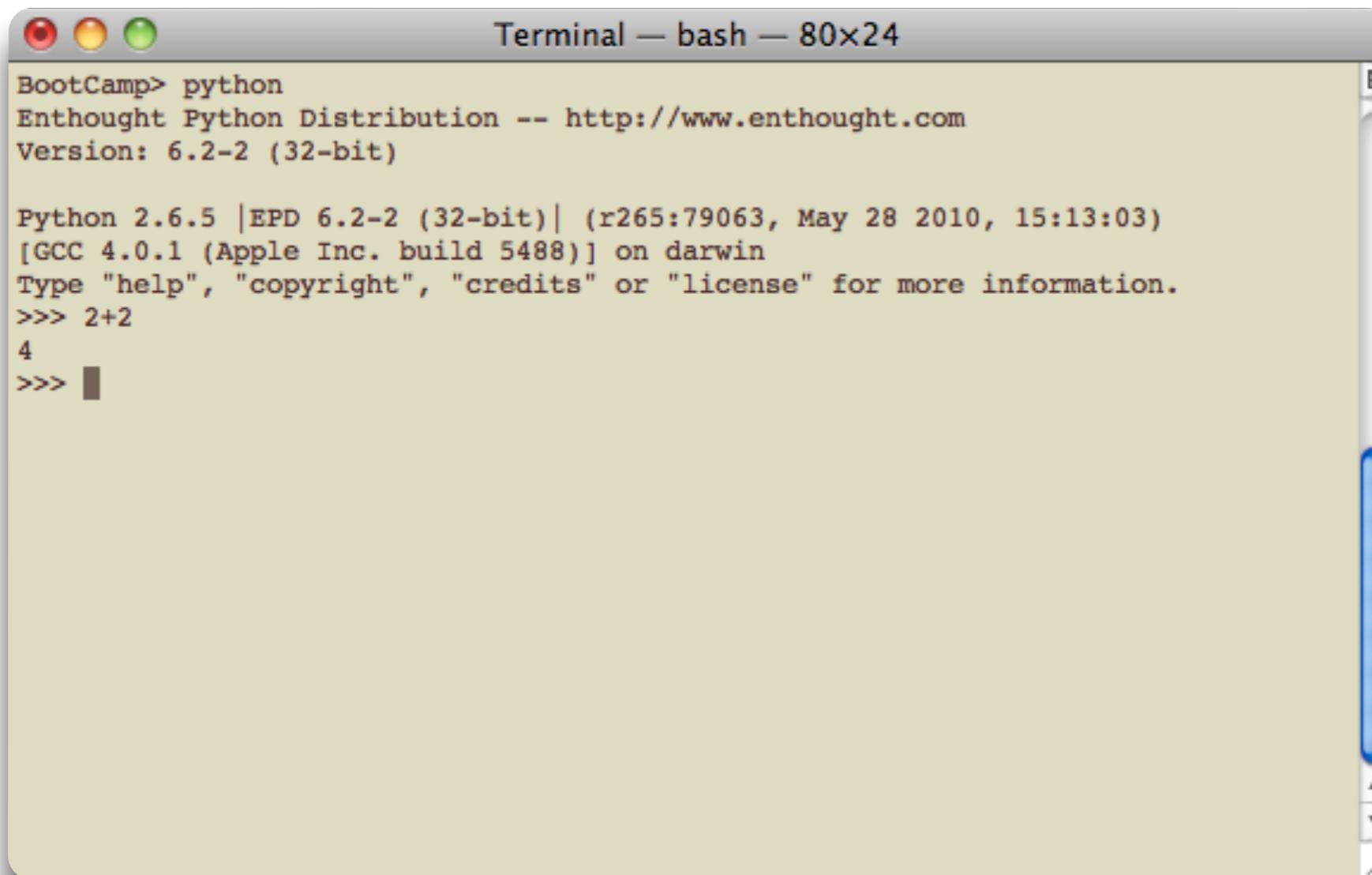


Animations



Ask Questions!!!

Firing up the interpreter



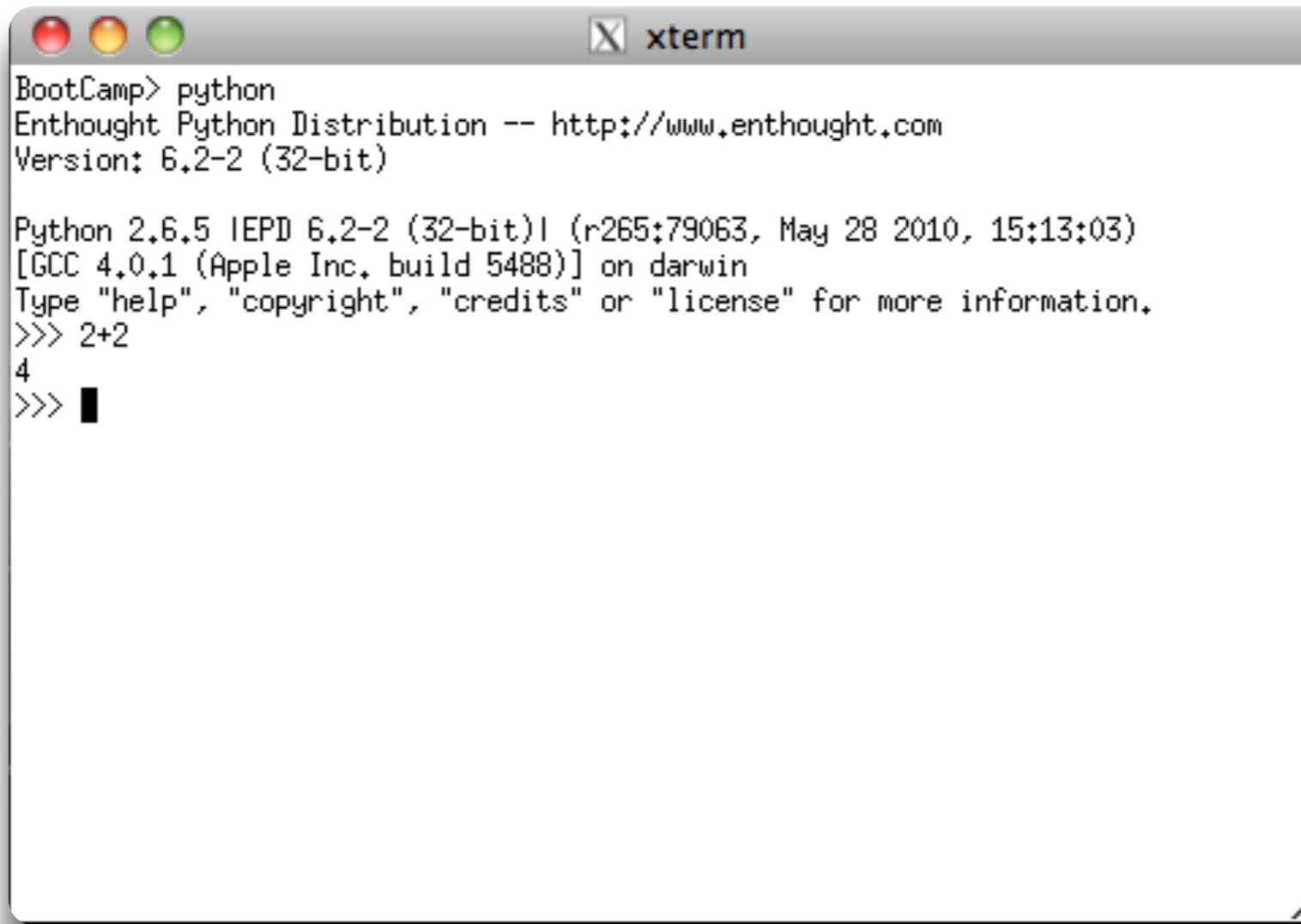
A screenshot of an OS X Terminal window titled "Terminal — bash — 80x24". The window shows the following text:

```
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
>>> 
```

OS X (terminal)

Firing up the interpreter

A screenshot of an xterm window titled "xterm". The window has a Mac OS X style title bar with red, yellow, and green buttons. The terminal displays the following text:

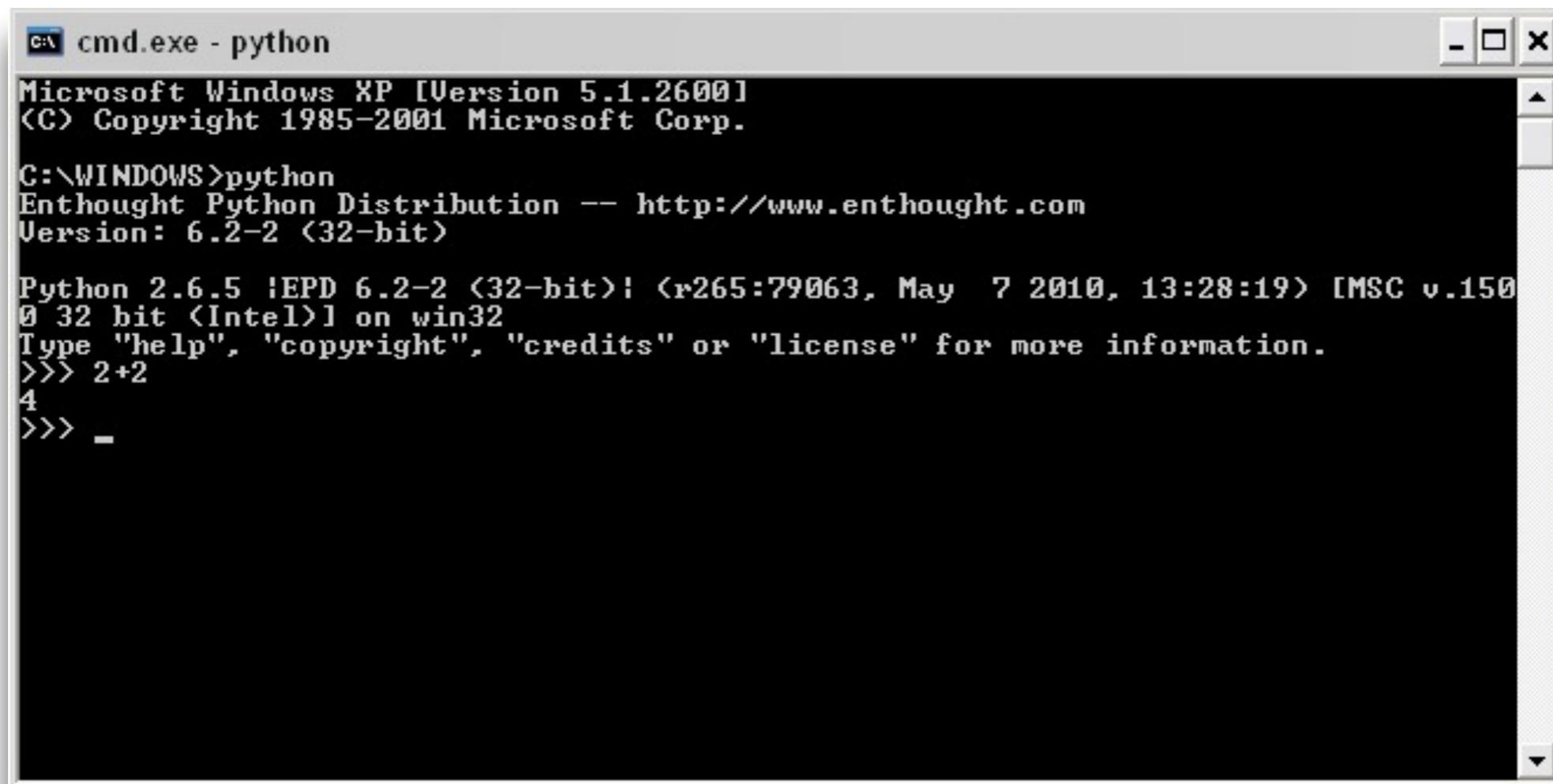
```
BootCamp> python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.2-2 (32-bit)

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
>>> ■
```

The window is centered on the screen with a slight shadow.

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

Firing up the interpreter



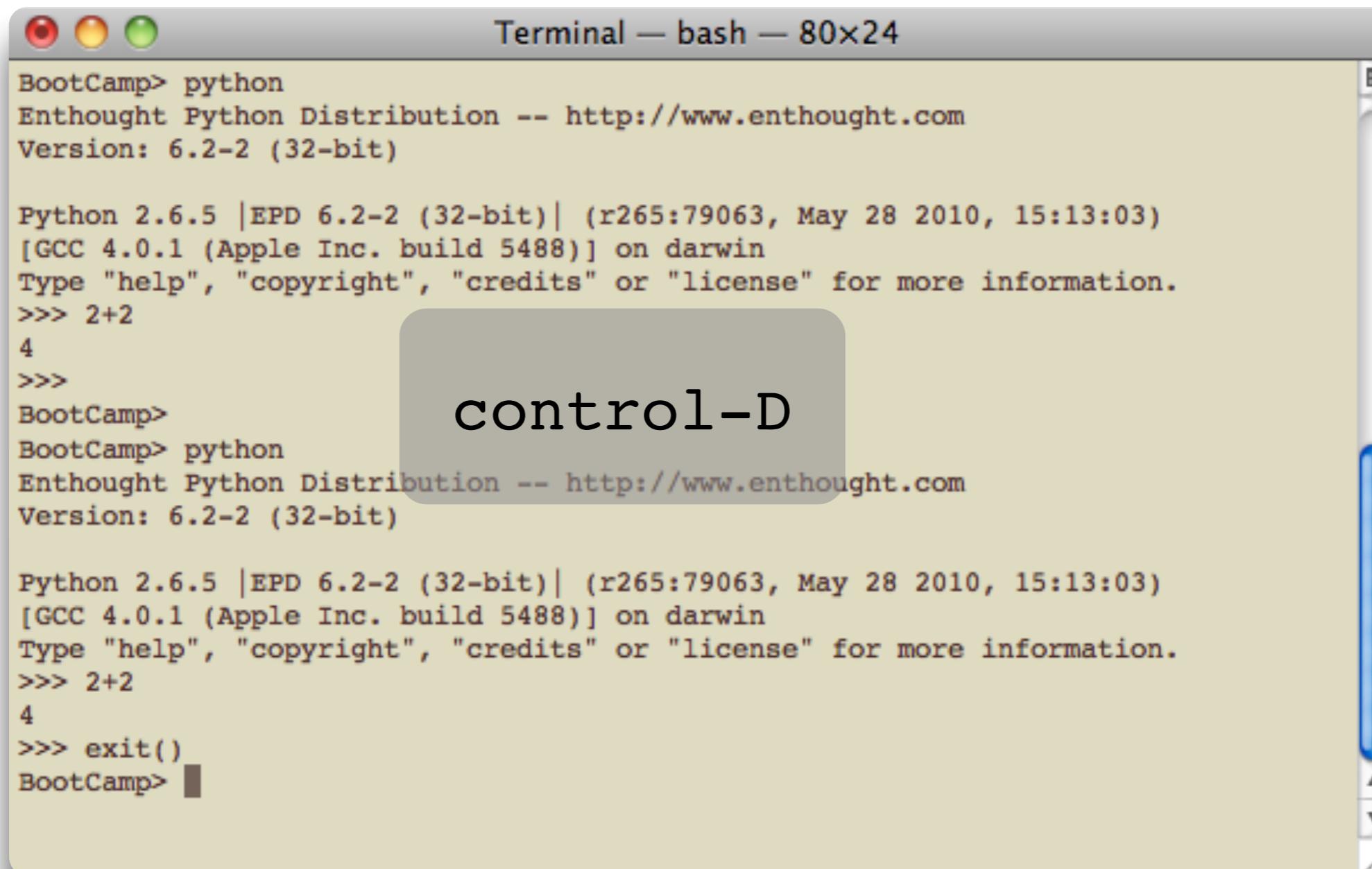
```
cmd.exe - python
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

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



```
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)

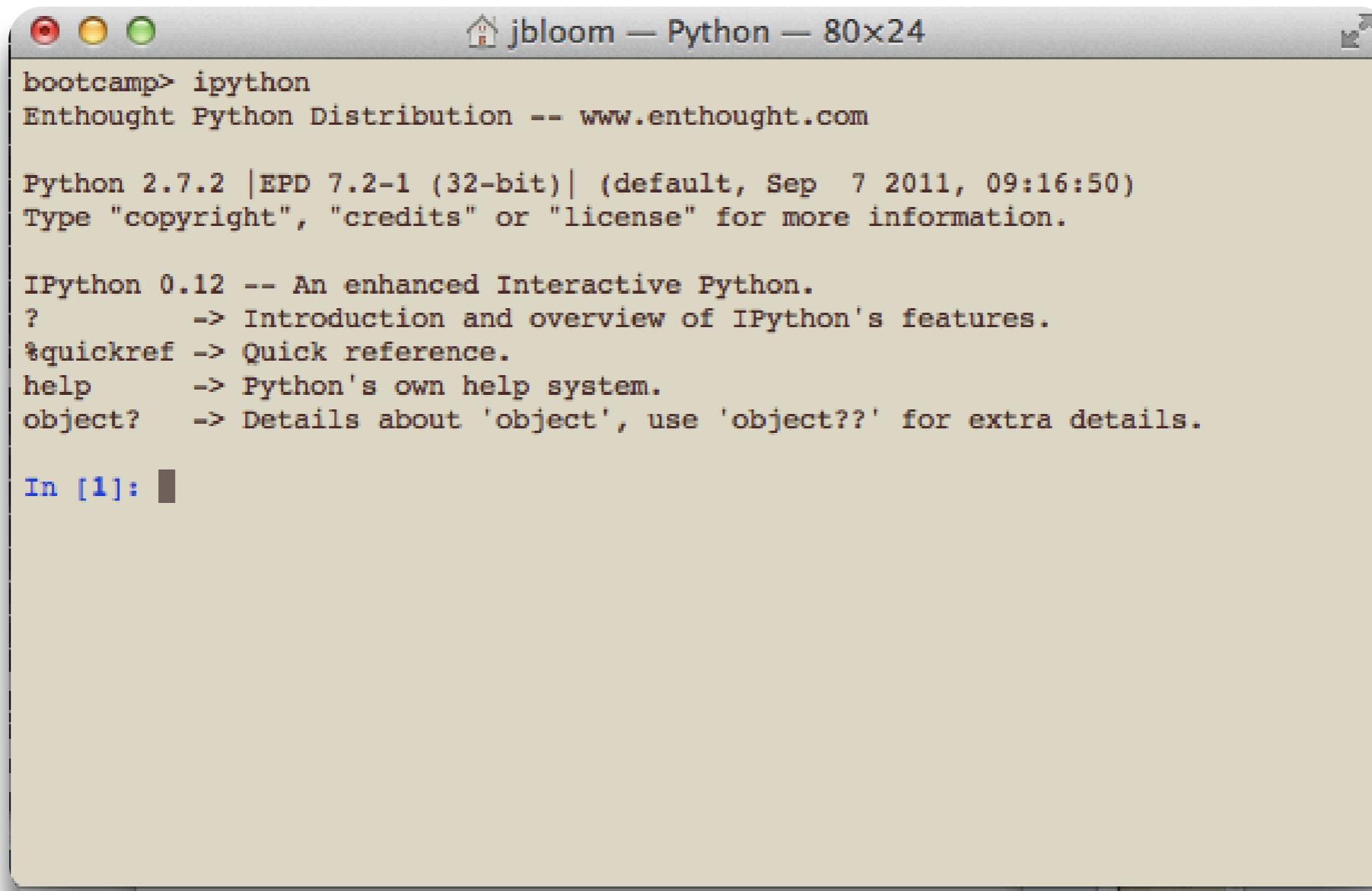
Python 2.6.5 |EPD 6.2-2 (32-bit)| (r265:79063, May 28 2010, 15:13:03)
[GCC 4.0.1 (Apple Inc. build 5488)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> 2+2
4
>>> exit()
BootCamp>
```

control-D

to exit: either control-D or exit()

Firing up the interpreter



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]:

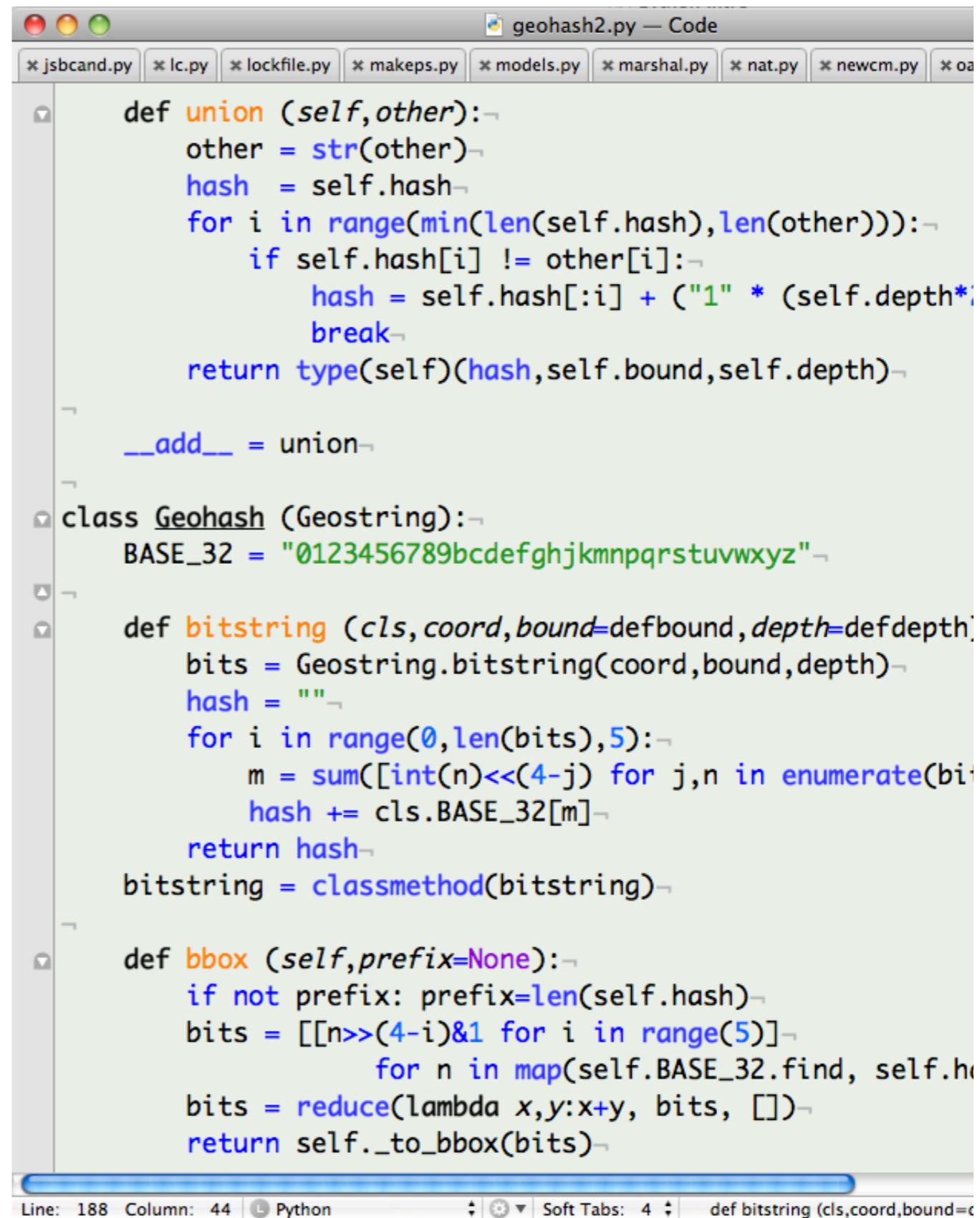
iPython

Editing Python Files



usually we name
python files with
a **.py** suffix

- New hotness
 - BBEdit, TextWrangler (Mac), NotePad++, SublimeText (windows), KWrite, Scribes, eggy (linux), Textmate
- Old standards
 - vim, emacs, nano, ect...



A screenshot of a Mac OS X application window titled "geohash2.py — Code". The window shows a Python script with syntax highlighting. The code defines a class `Geohash` which inherits from `Geostring`. It includes methods for union, addition, bitstring conversion, and bounding boxes. The code uses various Python features like list comprehensions, map, reduce, and lambda functions. The status bar at the bottom indicates the line and column numbers (Line: 188, Column: 44), the language (Python), and soft tabs settings.

```
def union (self,other):  
    other = str(other)  
    hash = self.hash  
    for i in range(min(len(self.hash),len(other))):  
        if self.hash[i] != other[i]:  
            hash = self.hash[:i] + ("1" * (self.depth*  
            break  
    return type(self)(hash,self.bound,self.depth)  
  
__add__ = union  
  
class Geohash (Geostring):  
    BASE_32 = "0123456789bcdefghjkmnpqrstuvwxyz"  
  
    def bitstring (cls,coord,bound=defbound,depth=defdepth):  
        bits = Geostring.bitstring(coord,bound,depth)  
        hash = ""  
        for i in range(0,len(bits),5):  
            m = sum([int(n)<<(4-j) for j,n in enumerate(bi  
            hash += cls.BASE_32[m]  
        return hash  
    bitstring = classmethod(bitstring)  
  
    def bbox (self,prefix=None):  
        if not prefix: prefix=len(self.hash)  
        bits = [[n>>(4-i)&1 for i in range(5)]  
                for n in map(self.BASE_32.find, self.h  
        bits = reduce(lambda x,y:x+y, bits, [])  
        return self._to_bbox(bits)
```

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
complied 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

```
>>> 6 / 5 ; 9 / 5 # integer division returns the floor
1
1
>>> 6 % 5 # mod operator
1
>>> 1 << 2 ## shift: move the number 1 by two bits to the left
          ##           that is make a new number 100 (base 2)
4
>>> 5 >> 1 ## shift: move the number 5 = 101 (base 2) one to
          ##           to the right (10 = 2)
2
>>> x = 2 ; y = 3 ## assign two variables on the same line!
>>> x | y          ## bitwise OR
3
>>> x ^ y          ## exclusive OR (10 ^ 11 = 01)
1
>>> x & y          ## bitwise AND
2
>>> x = x ^ y ; print x
1
>>> x += 3 ; print x
4
>>> x /= 2.0
>>> print x
2.0
```

We'll see a lot more mathy operators and functions later

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>()
      1
      2 TypeError: no ordering relation is defined for complex numbers
      3
      4
      5
      6
      7
      8
      9
     10
     11
     12
     13
     14
     15
     16
     17
     18
     19
     20
     21
     22
     23
     24
     25
     26
     27
     28
     29
     30
     31
     32
     33
     34
     35
     36
     37
     38
     39
     40
     41
     42
     43
     44
     45
     46
     47
     48
     49
     50
     51
     52
     53
     54
     55
     56
     57
     58
     59
     60
     61
     62
     63
     64
     65
     66
     67
     68
     69
     70
     71
     72
     73
     74
     75
     76
     77
     78
     79
     80
     81
     82
     83
     84
     85
     86
     87
     88
     89
     90
     91
     92
     93
     94
     95
     96
     97
     98
     99
     100
     101
     102
     103
     104
     105
     106
     107
     108
     109
     110
     111
     112
     113
     114
     115
     116
     117
     118
     119
     120
     121
     122
     123
     124
     125
     126
     127
     128
     129
     130
     131
     132
     133
     134
     135
     136
     137
     138
     139
     140
     141
     142
     143
     144
     145
     146
     147
     148
     149
     150
     151
     152
     153
     154
     155
     156
     157
     158
     159
     160
     161
     162
     163
     164
     165
     166
     167
     168
     169
     170
     171
     172
     173
     174
     175
     176
     177
     178
     179
     180
     181
     182
     183
     184
     185
     186
     187
     188
     189
     190
     191
     192
     193
     194
     195
     196
     197
     198
     199
     200
     201
     202
     203
     204
     205
     206
     207
     208
     209
     210
     211
     212
     213
     214
     215
     216
     217
     218
     219
     220
     221
     222
     223
     224
     225
     226
     227
     228
     229
     230
     231
     232
     233
     234
     235
     236
     237
     238
     239
     240
     241
     242
     243
     244
     245
     246
     247
     248
     249
     250
     251
     252
     253
     254
     255
     256
     257
     258
     259
     260
     261
     262
     263
     264
     265
     266
     267
     268
     269
     270
     271
     272
     273
     274
     275
     276
     277
     278
     279
     280
     281
     282
     283
     284
     285
     286
     287
     288
     289
     290
     291
     292
     293
     294
     295
     296
     297
     298
     299
     300
     301
     302
     303
     304
     305
     306
     307
     308
     309
     310
     311
     312
     313
     314
     315
     316
     317
     318
     319
     320
     321
     322
     323
     324
     325
     326
     327
     328
     329
     330
     331
     332
     333
     334
     335
     336
     337
     338
     339
     340
     341
     342
     343
     344
     345
     346
     347
     348
     349
     350
     351
     352
     353
     354
     355
     356
     357
     358
     359
     360
     361
     362
     363
     364
     365
     366
     367
     368
     369
     370
     371
     372
     373
     374
     375
     376
     377
     378
     379
     380
     381
     382
     383
     384
     385
     386
     387
     388
     389
     390
     391
     392
     393
     394
     395
     396
     397
     398
     399
     400
     401
     402
     403
     404
     405
     406
     407
     408
     409
     410
     411
     412
     413
     414
     415
     416
     417
     418
     419
     420
     421
     422
     423
     424
     425
     426
     427
     428
     429
     430
     431
     432
     433
     434
     435
     436
     437
     438
     439
     440
     441
     442
     443
     444
     445
     446
     447
     448
     449
     450
     451
     452
     453
     454
     455
     456
     457
     458
     459
     460
     461
     462
     463
     464
     465
     466
     467
     468
     469
     470
     471
     472
     473
     474
     475
     476
     477
     478
     479
     480
     481
     482
     483
     484
     485
     486
     487
     488
     489
     490
     491
     492
     493
     494
     495
     496
     497
     498
     499
     500
     501
     502
     503
     504
     505
     506
     507
     508
     509
     510
     511
     512
     513
     514
     515
     516
     517
     518
     519
     520
     521
     522
     523
     524
     525
     526
     527
     528
     529
     530
     531
     532
     533
     534
     535
     536
     537
     538
     539
     540
     541
     542
     543
     544
     545
     546
     547
     548
     549
     550
     551
     552
     553
     554
     555
     556
     557
     558
     559
     560
     561
     562
     563
     564
     565
     566
     567
     568
     569
     570
     571
     572
     573
     574
     575
     576
     577
     578
     579
     580
     581
     582
     583
     584
     585
     586
     587
     588
     589
     590
     591
     592
     593
     594
     595
     596
     597
     598
     599
     600
     601
     602
     603
     604
     605
     606
     607
     608
     609
     610
     611
     612
     613
     614
     615
     616
     617
     618
     619
     620
     621
     622
     623
     624
     625
     626
     627
     628
     629
     630
     631
     632
     633
     634
     635
     636
     637
     638
     639
     640
     641
     642
     643
     644
     645
     646
     647
     648
     649
     650
     651
     652
     653
     654
     655
     656
     657
     658
     659
     660
     661
     662
     663
     664
     665
     666
     667
     668
     669
     670
     671
     672
     673
     674
     675
     676
     677
     678
     679
     680
     681
     682
     683
     684
     685
     686
     687
     688
     689
     690
     691
     692
     693
     694
     695
     696
     697
     698
     699
     700
     701
     702
     703
     704
     705
     706
     707
     708
     709
     710
     711
     712
     713
     714
     715
     716
     717
     718
     719
     720
     721
     722
     723
     724
     725
     726
     727
     728
     729
     730
     731
     732
     733
     734
     735
     736
     737
     738
     739
     740
     741
     742
     743
     744
     745
     746
     747
     748
     749
     750
     751
     752
     753
     754
     755
     756
     757
     758
     759
     760
     761
     762
     763
     764
     765
     766
     767
     768
     769
     770
     771
     772
     773
     774
     775
     776
     777
     778
     779
     780
     781
     782
     783
     784
     785
     786
     787
     788
     789
     790
     791
     792
     793
     794
     795
     796
     797
     798
     799
     800
     801
     802
     803
     804
     805
     806
     807
     808
     809
     810
     811
     812
     813
     814
     815
     816
     817
     818
     819
     820
     821
     822
     823
     824
     825
     826
     827
     828
     829
     830
     831
     832
     833
     834
     835
     836
     837
     838
     839
     840
     841
     842
     843
     844
     845
     846
     847
     848
     849
     850
     851
     852
     853
     854
     855
     856
     857
     858
     859
     860
     861
     862
     863
     864
     865
     866
     867
     868
     869
     870
     871
     872
     873
     874
     875
     876
     877
     878
     879
     880
     881
     882
     883
     884
     885
     886
     887
     888
     889
     890
     891
     892
     893
     894
     895
     896
     897
     898
     899
     900
     901
     902
     903
     904
     905
     906
     907
     908
     909
     910
     911
     912
     913
     914
     915
     916
     917
     918
     919
     920
     921
     922
     923
     924
     925
     926
     927
     928
     929
     930
     931
     932
     933
     934
     935
     936
     937
     938
     939
     940
     941
     942
     943
     944
     945
     946
     947
     948
     949
     950
     951
     952
     953
     954
     955
     956
     957
     958
     959
     960
     961
     962
     963
     964
     965
     966
     967
     968
     969
     970
     971
     972
     973
     974
     975
     976
     977
     978
     979
     980
     981
     982
     983
     984
     985
     986
     987
     988
     989
     990
     991
     992
     993
     994
     995
     996
     997
     998
     999
     1000
```

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 stings (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>()
      1 TypeError: cannot concatenate 'str' and 'int' objects
      2 >>> print 'I want ' + str(3) + ' eggs and no ' + s
      3 I want 3 eggs and no spam
      4 >>> pi = 3.14159
      5 >>> print 'I want ' + str(pi) + ' eggs and no ' + s
      6 I want 3.14159 eggs and no spam
      7 >>> print str(True) + ":" + ' I want ' + str(pi) + ' eggs and no ' + s
      8 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

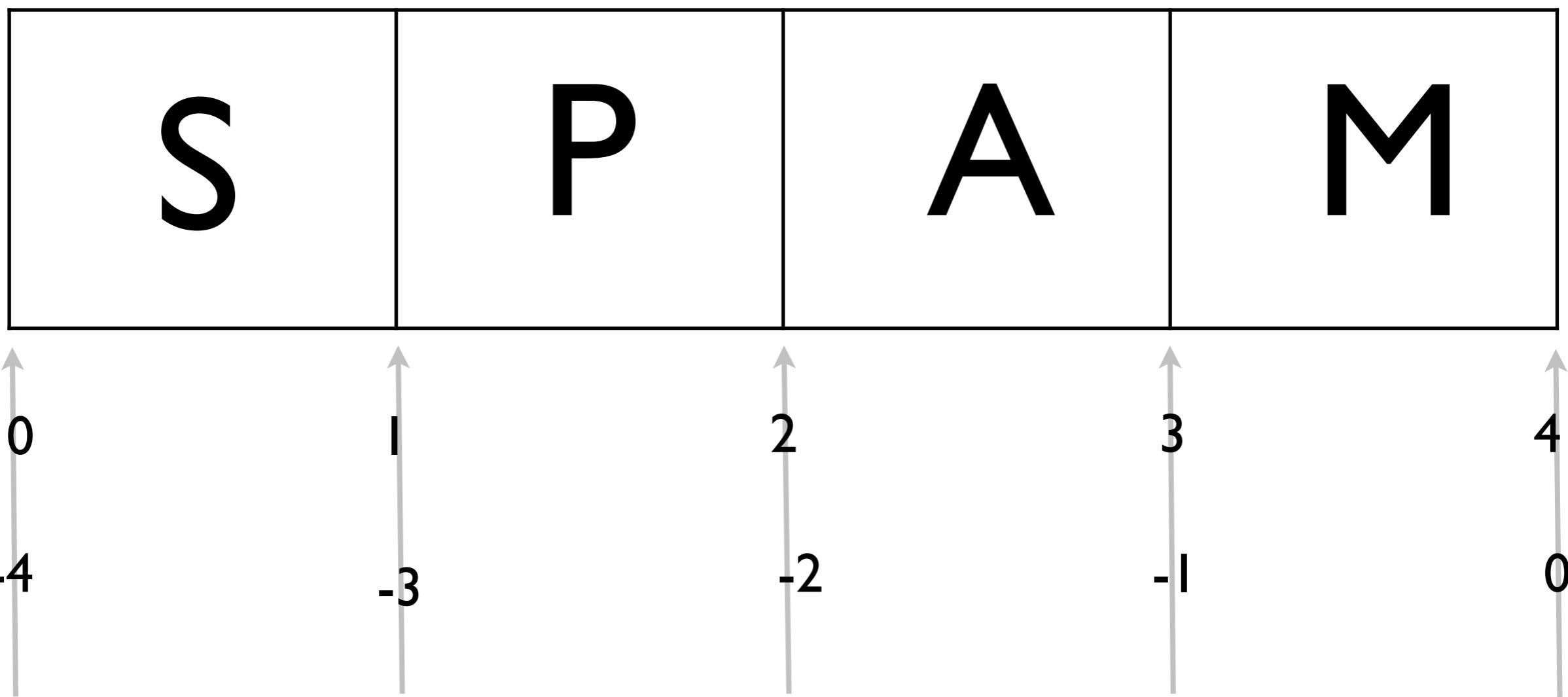
We can think of strings as arrays (although, unlike in C you never really need to deal with directly addressing characters locations in memory)

```
>>> 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

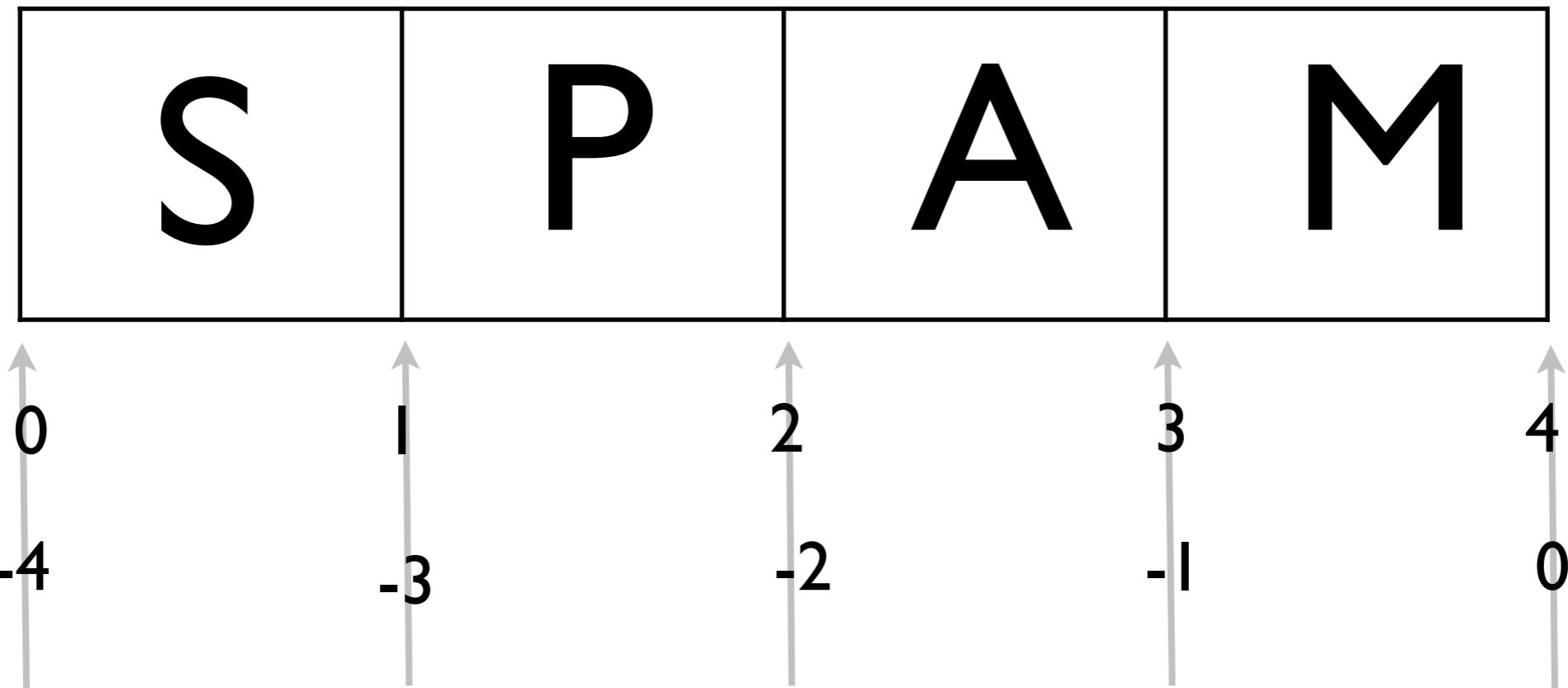
Strings

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'
```

$$s = s[:n] + s[n:] \text{ for all } n$$

Basic Control (Flow)

- Python has pretty much all what you use
 - `if`, `elif`, `else`, `for`, `while`, `break`, and `continue`
- Does not have:
 - `case` (explicity), `goto`
- Does have:
 - `pass`

Basic Control (Flow)

Flow is done within blocks (where indentation matters)

```
>>> x = 1  
>>> if x > 0:  
        print "yo"  
else:  
        print "dude"
```

yo

```
>>> x = 1  
>>> if x > 0:  
...     print "yo"  
... else:  
...     print "dude"  
...  
yoe:
```

looks like this
(note the ...)

Note colons and indentations (tabbed or spaced)

```
>>> x = 1  
>>> if x > 0:  
        print "yo"  
else:  
        print "dude"
```

yo

Indentation within the same block must be the same but not
within different blocks (thought this is ugly)

Basic Control (Flow)

one-liners

```
>>> print "yo" if x > 0 else "dude"  
"dude"
```

a small program...

```
>>> x = 1  
>>> while True:  
    print "yo" if x > 0 else "dude"  
    x *= -1  
  
yo  
dude  
yo  
dude  
...  
yo  
dude  
yo  
^C  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
KeyboardInterrupt
```



Control-C usually
drops you back to the
prompt

Basic Control (Flow)

Blocks cannot be empty

```
>>> x = "fried goldfish"
>>> if x == "spam for dinner":
    print "I will destroy the universe"
else:
    # I'm fine with that. I'll do nothing

File "<stdin>", line 5

    ^
IndentationError: expected an indented block
>>>
```

pass is a "do nothing" statement

```
>>> if x == "spam for dinner":
    print "I will destroy the universe"
else:
    # I'm fine with that. I'll do nothing
pass

>>>
```

```
# set some initial variables. Set the initial temperature low
faren = -1000

# we dont want this going on forever, let's make sure we cannot have too many attempts
max_attempts = 6
attempt = 0

while faren < 100:
    # let's get the user to tell us what temperature it is
    newfaren = float(raw_input("Enter the temperature (in Fahrenheit): "))
    if newfaren > faren:
        print "It's getting hotter"
    elif newfaren < faren:
        print "It's getting cooler"
    else:
        # nothing has changed, just continue in the loop
        continue
    faren = newfaren # now set the current temp to the new temp just entered
    attempt += 1 # bump up the attempt number
    if attempt >= max_attempts:
        # we have to bail out
        break

if attempt >= max_attempts:
    # we bailed out because of too many attempts
    print "Too many attempts at raising the temperature."
else:
    # we got here because it's hot
    print "it's hot here, man."
```

```
BootCamp> python temp1.py
Enter the temperature (in Fahrenheit): 1
It's getting hotter
Enter the temperature (in Fahrenheit): 2
It's getting hotter
Enter the temperature (in Fahrenheit): 3
It's getting hotter
Enter the temperature (in Fahrenheit): 4
It's getting hotter
Enter the temperature (in Fahrenheit): -1
It's getting cooler
Enter the temperature (in Fahrenheit): 10
It's getting hotter
Too many attempts at raising the temperature.
BootCamp>
```

```
BootCamp> python temp1.py
Enter the temperature (in Fahrenheit): 3
It's getting hotter
Enter the temperature (in Fahrenheit): -45
It's getting cooler
Enter the temperature (in Fahrenheit): 101
It's getting hotter
it's hot here, man.
BootCamp>
```

```
# set some initial variables. Set the initial temperature low
faren = -1000

# we dont want this going on forever, let's make sure we cannot have too many attempts
max_attempts = 6
attempt = 0

while faren < 100 and (attempt < max_attempts):
    # let's get the user to tell us what temperature it is
    newfaren = float(raw_input("Enter the temperature (in Fahrenheit): "))
    if newfaren > faren:
        print "It's getting hotter"
    elif newfaren < faren:
        print "It's getting cooler"
    else:
        # nothing has changed, just continue in the loop
        continue
    faren = newfaren # now set the current temp to the new temp just entered
    attempt += 1 # bump up the attempt number

if attempt >= max_attempts:
    # we bailed out because of too many attempts
    print "Too many attempts at raising the temperature."
else:
    # we got here because it's hot
    print "it's hot here, man."
```

Exercise for the Breakout

Write a program which allows the user to build up a sentence one word at a time, stopping when they enter a period (.), an exclamation (!), or a question mark (?)

Example interaction:

```
Please enter a word in the sentence (enter . ! or ? to end.): My
...currently: My
Please enter a word in the sentence (enter . ! or ? to end.): name
...currently: My name
Please enter a word in the sentence (enter . ! or ? to end.): is
...currently: My name is
Please enter a word in the sentence (enter . ! or ? to end.): Slim
...currently: My name is Slim
Please enter a word in the sentence (enter . ! or ? to end.): Shady
...currently: My name is Slim Shady
Please enter a word in the sentence (enter . ! or ? to end.): !
--->My name is Slim Shady!
```