Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

Introduction to Python 2 for statisticians

Will Landau

Iowa State University

November 18, 2013

Outline

Basic elements

The NumPy module

Other useful modules

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

Outline

Basic elements

The NumPy module

Other useful modules

Preliminaries

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

- ▶ Python is a high-level multipurpose interpreted language.
- ▶ It's clumsier than R at statistical number crunching, but better than R at string manipulation.
- ▶ Open the interpreter by typing python into the command line.

```
1 > python
2 Python 2.7.2 (default, Oct 11 2012, 20:14:37)
3 [GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang −418.0.60)]
      on darwin
4 Type "help", "copyright", "credits" or "license" for more
      information.
5 >>>
```

Preliminaries

► You can create variables and do arithmetic:

```
6  >>> a = 1
7  >>> b = 2
8  >>> c = "goober"
9  >>> c
10 'goober'
11 >>> a+b
12 3
13 >>>
```

► You can make a python script like world.py:

```
14 s = "Hello World"
15 print(s)
```

► And run it in the command line:

```
16 > python hello_world.py
17 Hello World
```

Preliminaries

▶ Use the hash sign for single-line comments.

```
18  >>> # print("Hello world!")
19  >>>
```

▶ Triple-quoted strings span multiple lines and serve as
multi-line comments.

```
20  # a.py
21  """
22  This program
23  does nothing.
24  """
```

```
25  > python a.py
26  >
```

Preliminaries

- ▶ Formatting strings

```
27  >>> s = "Today's date is {month}/{day}/{year}".format(month = 10, day =
        22, \
28  ...          year = 2012)
29  >>>
30  ... print(s)
31  Today's date is 10/22/2012
32  >>>
```

- ▶ Every string has builtin methods such as format(),
  which can be accessed with the dot operator.

- ▶ There are multiple ways to format output.

```
33  >>> a = 3
34  >>> b = 4.8878
35  >>> s = format("sample %d: mass= %0.3fg" % (a, b))
36  >>> print(s)
37  sample 3: mass= 4.888g
38  >>>
39  >>> print("sample %d: mass= %0.3fg" % (a, b))
40  sample 3: mass= 4.888g
41  >>>
```

User-defined functions

► I can define and use my own function like this:

```
42 >>> def f1(a):
43 ...     if a == 0:
44 ...         print("hi")
45 ...         return(0)
46 ...     elif a < 0:
47 ...         print("stop")
48 ...         return(1)
49 ...     else:
50 ...         return(5)
51 ...
52 >>> f1(0)
53 hi
54 0
55 >>> f1(1)
56 5
57 >>> f1(-1)
58 stop
59 1
60 >>>
```

Indentation

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

- ▶ In python, indention is used to denote nested blocks of code (like { and } in C). Indentation must be consistent.
- ▶ The following script, a.py, has an indentation error.

```
61  # a.py
62  def f1(a):
63    if a == 0:
64      print("hi")
65      return(0)
66    elif a < 0:
67      print("stop")
68      return(1)
69    else:
70    return(5)
```

- ▶ When I try to run it,

```
71  > python a.py
72    File "a.py", line 10
73      return(5)
74           ^
75  IndentationError: expected an indented block
```

Line continuation

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ With the exception of multi-line quotes, you have to use
the line continuation character, '\', when you want to
wrap text in your code:

```
76  >>> 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
77  29
78  >>> 1+1+1+1+1+1+1+
79    File "<stdin>", line 1
80      1+1+1+1+1+1+1+
81                   ^
82  SyntaxError: invalid syntax
83  >>>
84  >>> 1+1+1+1+1+1+1+ \
85  ... 1+1+1+1+1+1+1+1
86  15
87  >>>
```

Logic and control flow

```
88  >>> 1 and 2
89  2
90  >>> 1 == 1
91  True
92  >>> 1 == 0
93  False
94  >>> 1 == 1 and 2 == 0
95  False
96  >>> 1 > 1 or 2 <= 5
97  True
98  >>> not True
99  False
100 >>> True and not False
101 True
102 >>> if True:
103 ...     print("yes")
104 ... else:
105 ...     print("no")
106 ...
107 yes
108 >>>
```

Logic and control flow

```
109  >>> a = 1
110  >>> if a == 2:
111  ...     print("two")
112  ... elif a < -1000:
113  ...     print("a is small")
114  ... elif a > 100 and not a % 2:
115  ...     print("a is big and even")
116  ... else:
117  ...     print("a is none of the above.")
118  ...
119  a is none of the above.
120  >>>
```

Stings: where Python is strong

▶ You can use single, double, or triple quotes to denote string literals.

```
121 >>> a = "Hello World"
122 >>> b = 'Python is groovy'
123 >>> c = """Computer says 'No'"""
124 >>>
```

▶ Triple quotes can extend over multiple lines, but single and double quotes cannot.

```
125 >>> c = """Computer says 'no'
126 ... because another computer
127 ... says yes"""
128 >>> a = "hello
129     File "<stdin>", line 1
130         a = "hello
131                   ^
132 SyntaxError: EOL while scanning string literal
133 >>>
```

Strings: where Python is strong

▶ Strings are stored as sequences of characters.

```
134 >>> a = "Hello World"
135 >>> a[0]
136 'H'
137 >>> a[:5]
138 'Hello'
139 >>> a[6:]
140 'World'
141 >>> a[3:8]
142 'lo Wo'
143 >>>
```

▶ You can convert numeric types into strings and vice versa.

```
144 >>> z = "90"
145 >>> z
146 '90'
147 >>> int(z)
148 90
149 >>> float(z)
150 90.0
151 >>> str(90.25)
152 '90.25'
153 >>>
```

Strings: where Python is strong

▶ And you can concatenate strings.

```
154 >>> "123" + "abc"
155 '123abc'
156 >>> "123" + str(123.45)
157 '123123.45'
158 >>> a = 1
159 >>> b = "2"
160 >>> str(a) + b
161 '12'
162 >>>
```

Strings: where Python is strong

▶ There are several useful methods for strings.

```
163 >>> s = "Hello world!"
164 >>> len(s)
165 12
166 >>>
167 >>> s = "5, 4, 2, 9, 8, 7, 28"
168 >>> s.count(",")
169 6
170 >>> s.find("9, ")
171 9
172 >>> s[9:12]
173 '9, '
174 >>> "abc123".isalpha()
175 False
176 >>> "abc123".isalnum()
177 True
178 >>> s.split(",")
179 ['5', ' 4', ' 2', ' 9', ' 8', ' 7', ' 28']
180 >>> ", ".join(["ready", "set", "go"])
181 'ready, set, go'
182 >>> "ready\n set\n go".splitlines()
183 ['ready', ' set', ' go']
184 >>> "ready set go".splitlines()
185 ['ready set go']
186 >>>
```

Lists

▶ In python, a list is an ordered sequence of objects, each
of which can have any type.

```
187  >>> s = [1, 2, "Five!", ["Three, sir!", "Three!"]]
188  >>> len(s)
189  4
190  >>>
191  >>> s[0:1]
192  [1]
193  >>> s[2]
194  'Five!'
195  >>> s[2][1]
196  'i'
197  >>> s[3]
198  ['Three, sir!', 'Three!']
199  >>> s[3][0]
200  'Three, sir!'
201  >>> s[3][1]
202  'Three!'
203  >>> s[3][1][1]
204  'h'
205  >>> s.append("new element")
206  >>> s
207  [1, 2, 'Five!', ['Three, sir!', 'Three!'], 'new element']
```

Lists

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ I can append and remove list elements.

```
208 >>> l = ["a", "b", "c"]
209 >>> l.append("d")
210 >>> l.append("c")
211 >>> l
212 ['a', 'b', 'c', 'd', 'c']
213 >>> l.remove("a")
214 >>> l
215 ['b', 'c', 'd', 'c']
216 >>> l.remove("c")
217 >>> l
218 ['b', 'd', 'c']
219 >>> l.remove("c")
220 >>> l
221 ['b', 'd']
222 >>>
```

Tuples

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

```
223  >>> a = ()
224  >>> b = (3,)
225  >>> c = (3,4,"thousand")
226  >>> len(c)
227  3
228  >>>
229  >>> number1, number2, word = c
230  >>> number1
231  3
232  >>> number2
233  4
234  >>> word
235  'thousand'
236  >>> keys =["name", "status", "ID"]
237  >>> values = ["Joe", "approved", 23425]
238  >>> z = zip(keys, values)
239  >>> z
240  [('name', 'Joe'), ('status', 'approved'), ('ID', 23425)]
```

Dictionaries

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

```
241 >>> stock = {
242 ... "name" : "GOOG",
243 ... "shares" : 100,
244 ... "price" : 490.10 }
245 >>> stock
246 {'price': 490.1, 'name': 'GOOG', 'shares': 100}
247 >>> stock["name"]
248 'GOOG'
249 >>> stock["date"] = "today"
250 >>> stock
251 {'date': 'today', 'price': 490.1, 'name': 'GOOG', 'shares': 100}
252 >>> keys = ["name", "status", "ID"]
253 >>> values = ["Joe", "approved", 23425]
254 >>> zip(keys, values)
255 [('name', 'Joe'), ('status', 'approved'), ('ID', 23425)]
256 >>> d = dict(zip(keys, values))
257 >>> d
258 {'status': 'approved', 'name': 'Joe', 'ID': 23425}
259 >>>
```

Iteration and looping

- ▶ There are many ways to iterate.

```
260  # a.py
261  a = "Hello World"
262  # Print out the individual characters in a
263  for c in a:
264    print c
```

```
265  > python a.py
266  H
267  e
268  l
269  l
270  o
271
272  W
273  o
274  r
275  l
276  d
```

Iteration and looping

```
277  # a.py
278  b = ["Dave","Mark","Ann","Phil"]
279  # Print out the members of a list
280  for name in b:
281      print name
```

```
282  > python a.py
283  Dave
284  Mark
285  Ann
286  Phil
```

```
287  # a.py
288  c = { 'GOOG' : 490.10, 'IBM' : 91.50, 'AAPL' : 123.15 }
289  # Print out all of the members of a dictionary
290  for key in c:
291      print key, c[key]
```

```
292  > python a.py
293  GOOG 490.1
294  AAPL 123.15
295  IBM 91.5
```

Iteration and looping

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

```
296  # a.py
297  for n in [0, 1,2,3,4,5,6,7,8,9]:
298    print("2 to the %d power is %d" % (n, 2**n))
```

```
299  > python a.py
300  2 to the 0 power is 1
301  2 to the 1 power is 2
302  2 to the 2 power is 4
303  2 to the 3 power is 8
304  2 to the 4 power is 16
305  2 to the 5 power is 32
306  2 to the 6 power is 64
307  2 to the 7 power is 128
308  2 to the 8 power is 256
309  2 to the 9 power is 512
```

```
310  # a.py
311  for n in range(9):
312      print("2 to the %d power is %d" % (n, 2**n))
```

```
313  > python a.py
314  2 to the 0 power is 1
315  2 to the 1 power is 2
316  2 to the 2 power is 4
317  2 to the 3 power is 8
318  2 to the 4 power is 16
319  2 to the 5 power is 32
320  2 to the 6 power is 64
321  2 to the 7 power is 128
322  2 to the 8 power is 256
323  2 to the 9 power is 512
```

range() and xrange()

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ For lengthy iterations, don't use range() because it
fully populates a list and takes up a lot of memory.

▶ Instead, use xrange(), which gives you your iteration
indices on a need-to-know basis.

```
324  # a.py
325  x = 0
326  for n in xrange(999999999):
327    x = x + 1
328  print(x)
```

Generators

- ▶ range() is a special case of a larger class of functions called generators.

```
329  >>> def countdown(n):
330  ...     print "Counting down!"
331  ...     while n > 0:
332  ...         yield n # Generate a value (n)
333  ...         n -= 1
334  ...
335  >>> c = countdown(5)
336  >>> c.next()
337  Counting down!
338  5
339  >>> c.next()
340  4
341  >>> c.next()
342  3
343  >>>
```

List comprehensions

```
344 >>> nums = [1, 2, 3, 4, 5]
345 >>> squares = [n * n for n in nums]
346 >>> squares
347 [1, 4, 9, 16, 25]
348 >>> a = [-3,5,2,-10,7,8]
349 >>> b = 'abc'
350 >>> [2*s for s in a]
351 [-6, 10, 4, -20, 14, 16]
352 >>> [s for s in a if s >= 0]
353 [5, 2, 7, 8]
354 >>> [(x,y) for x in a
355 ...    for y in b if x > 0 ]
356 [(5, 'a'), (5, 'b'), (5, 'c'), (2, 'a'), (2, 'b'), (2, 'c'), (7, 'a'),
357 (7, 'b'), (7, 'c'), (8, 'a'), (8, 'b'), (8, 'c')]
358 >>> [(1,2), (3,4), (5,6)]
359 [(1, 2), (3, 4), (5, 6)]
```

▶ General syntax:

```
360 [expression for item_1 in iterable_1 if condition_1
361              for item_2 in iterable_2 if condition_2
362              ...
363              for item_N in iterable_N if condition_N ]
```

Lambda functions, `filter()`, `map()`, and `reduce()`

- ▶ **Lambda function**: a compact way of writing a function. You can think of a lambda function as a "function literal".
- ▶ `filter(fun, list)`: returns a list of all the elements e in list for which fun(e) is true.
- ▶ `map(fun, list)`: applies fun to each element of list and returns the result in a new list.
- ▶ `reduce(fun, list)`: equivalent to the following (length of list is n).

```
364  value = fun(list[0], list[1])
365  value = fun(value, list[2])
366  value = fun(value, list[3])
367  ...
368  value = fun(value, list[n])
```

Lambda functions, `filter()`, `map()`, and `reduce()`

```
369  >>> f = lambda x: x > 3 and x % 2 != 0
370  >>> f(4)
371  False
372  >>> f(5)
373  True
374  >>> f(6)
375  False
376  >>>
377  >>> filter(lambda x: x > 3, [0, 1, 2, 3, 4, 5])
378  [4, 5]
379  >>>
380  >>>
381  >>> l = range(3)
382  >>> map(str, l)
383  ['0', '1', '2']
384  >>>
385  >>> map(lambda x: x*x, l)
386  [0, 1, 4]
387  >>>
388  >>> reduce(lambda x, y: x+y, range(1, 11)) # sum the numbers 1 to 10
389  55
390  >>>
```

File I/O

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ If I run:

```
391  # a.py
392  import random
393
394  f = open("data.txt", "w")
395  f.write("x y\n")
396  for i in xrange(10):
397    f.write("%0.3f %0.3f\n" % (random.random(), random.random()))
```

▶ The file, data.txt, is generated:

```
398  x y
399  0.506 0.570
400  0.887 0.792
401  0.921 0.641
402  0.894 0.664
403  0.494 1.000
404  0.745 0.734
405  0.274 0.127
406  0.075 0.381
407  0.449 0.995
408  0.355 0.807
```

File I/O

▶ I can read `data.txt` with:

```
409 >>> f = open("data.txt")
410 >>> header = f.readline()
411 >>> data = f.readlines()
412 >>>
413 >>> header
414 'x y\n'
415 >>> data
416 ['0.506 0.570\n', '0.887 0.792\n', '0.921 0.641\n', '0.894 0.664\n', '
        0.494 1.000\n',
417  '0.745 0.734\n', '0.274 0.127\n', '0.075 0.381\n', '0.449 0.995\n', '
        0.355 0.807\n']
418 >>>
419 >>> header = header.replace("\n","")
420 >>> header
421 'x y'
422 >>>
423 >>> d = [d.replace("\n","") for d in data]
424 >>> d
425 ['0.506 0.570', '0.887 0.792', '0.921 0.641', '0.894 0.664', '0.494
        1.000',
426  '0.745 0.734', '0.274 0.127', '0.075 0.381', '0.449 0.995', '0.355
        0.807']
427 >>>
```

File I/O

▶ And then I can process it into a nicer format.

```
428 >>> data = [d.split(" ") for d in data]
429 >>> data
430 [['0.506', '0.570'], ['0.887', '0.792'], ['0.921', '0.641'], ['0.894', '
        0.664'],
431 ['0.494', '1.000'], ['0.745', '0.734'], ['0.274', '0.127'], ['0.075', '
        0.381'],
432 ['0.449', '0.995'], ['0.355', '0.807']]
433 >>>
434 >>> data = [map(float, d) for d in data]
435 >>> data
436 [[0.506, 0.57], [0.887, 0.792], [0.921, 0.641], [0.894, 0.664], [0.494,
        1.0],
437 [0.745, 0.734], [0.274, 0.127], [0.075, 0.381], [0.449, 0.995], [0.355,
        0.807]]
```

Modules

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ Modules are external packages of code. The are not usually builtin, but they can be imported.

```
438  >>> sqrt(10)
439  Traceback (most recent call last):
440    File "<stdin>", line 1, in <module>
441  NameError: name 'sqrt' is not defined
442  >>>
443  >>> import math
444  >>> sqrt(10)
445  Traceback (most recent call last):
446    File "<stdin>", line 1, in <module>
447  NameError: name 'sqrt' is not defined
448  >>>
449  >>> math.sqrt(10)
450  3.1622776601683795
451  >>>
```

Modules

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

- ▶ If you don't want to write `math.sqrt()` every single time you want to compute a square root, you can use a shortcut.

```
452 >>> import math as m
453 >>> m.sqrt(10)
454 3.1622776601683795
455 >>>
```

- ▶ Or better yet,

```
456 >>> from math import *
457 >>> sqrt(10)
458 3.1622776601683795
459 >>>
```

- ▶ Where is the math module?

```
460 >>> import math
461 >>> math.__file__
462 '/usr/lib64/python2.6/lib-dynload/mathmodule.so'
463 >>>
```

Installing modules locally in impact1

▶ We don't have permission to install modules ourselves in /usr/. We have to either bother the STAT IT people or install our package locally.

▶ Suppose I want to install the SQLAlchemy module. To download it into my home directory, I work from linux command line on impact1.

```
 1 > cd ~
 2 > pwd
 3 /home/landau
 4 > ls
 5 stuff
 6 > wget http://prdownloads.sourceforge.net/sqlalchemy/SQLAlchemy−0.7.9.
        tar.gz?download
 7 # output of wget ...
 8 > ls
 9 stuff SQLAlchemy−0.7.9.tar.gz
10 > tar −zxvf SQLAlchemy−0.7.9.tar.gz
11 # output of tar...
12 > cd SQLAlchemy−0.7.9
13 > python setup.py build
14 # output of python...
15 > python setup.py install −−user
16 # output of python...
17 Installed /home/landau/.local/lib/python2.6/site−packages/SQLAlchemy
        −0.7.9−py2.6−linux−x86_64.egg
18 Processing dependencies for SQLAlchemy==0.7.9
19 Finished processing dependencies for SQLAlchemy==0.7.9
```

Installing modules locally in impact1

- ▶ IMPORTANT: take note that SQLAlchemy was installed in the directory, /home/landau/.local/lib/python2.6/site-packages/. I must "export" this path in my .bashrc file so that Python knows where my new module lives.
- ▶ I move to my home directory and open .bashrc:

```
20 [landau@impact1 SQLAlchemy−0.7.9]$ cd ˜
21 [landau@impact1 ˜]$ emacs .bashrc
```

- ▶ The file itself currently looks like this:

```
1 # .bashrc
2
3 # Source global definitions
4 if [ −f /etc/bashrc ]; then
5     . /etc/bashrc
6 fi
7
8 # User specific aliases and functions
```

Installing modules locally in impact1

- ▶ I add a couple lines to the end so that Python knows where to find my package:

```
 9 # .bashrc
10
11 # Source global definitions
12 if [ −f /etc/bashrc ]; then
13         . /etc/bashrc
14 fi
15
16 # User specific aliases and functions
17
18 export PYTHONPATH=$HOME/.local/lib/python2.6/site−packages:$PYTHONPATH
19 export PATH=$HOME/.local/bin:$PATH
```

- ▶ Once I've made the changes and I log out and in so that the changes take effect, I'm ready to import SQLAlchemy.

```
1 >>> import sqlalchemy
2 >>>
```

- ▶ Remember to use all lower case letters for libraries in the `import` statement.

The sys module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

- sys is a module of system-specific parameters and
  functions.

```
3  # a.py
4  import sys
5
6  for arg in sys.argv:
7    print arg
```

```
8  > python a.py 1 2 3 4 5 3sir! 3!
9  a.py
10 1
11 2
12 3
13 4
14 5
15 3sir!
16 3!
```

Outline

Basic elements

The NumPy module

Other useful modules

The NumPy module

► Important module for arrays and matrices.

```
17 >>> from numpy import *
18 >>> a = arange(15)
19 >>> a
20 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
21 >>> a = a.reshape(3,5)
22 >>> a
23 array([[ 0,  1,  2,  3,  4],
24        [ 5,  6,  7,  8,  9],
25        [10, 11, 12, 13, 14]])
26 >>> a.transpose()
27 >>> a.transpose()
28 array([[ 0,  5, 10],
29        [ 1,  6, 11],
30        [ 2,  7, 12],
31        [ 3,  8, 13],
32        [ 4,  9, 14]])
33 >>> a.shape
34 (3, 5)
35 >>> a.size
36 15
37 >>> type(a)
38 <type 'numpy.ndarray'>
39 >>>
40 >>> zeros( (3,4) )
41 array([[0., 0., 0., 0.],
42        [0., 0., 0., 0.],
43        [0., 0., 0., 0.]])
44 >>>
```

The `NumPy` module

```
45 >>> ones ( (2,3,4), dtype=int16 )                    # dtype can also be
        specified
46 array ([[[ 1, 1, 1, 1],
47          [ 1, 1, 1, 1],
48          [ 1, 1, 1, 1]],
49         [[ 1, 1, 1, 1],
50          [ 1, 1, 1, 1],
51          [ 1, 1, 1, 1]]], dtype=int16)
52 >>>
53 >>> empty ( (2,3) )
54 array ([[ 3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
55         [ 5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
56 >>>
57 >>> b = array ( [ [1.5,2,3], [4,5,6] ] )
58 >>> b
59 array ([[ 1.5,  2. ,  3. ],
60         [ 4. ,  5. ,  6. ]])
61 >>> print (b)
62 [[ 1.5  2.   3. ]
63  [ 4.   5.   6. ]]
64 >>>
65 >>> sum (b)
66 21.5
67 >>>
68 >>> a = array ( [20,30,40,50] )
69 >>> b = arange ( 4 )
70 >>> b
71 array ([0, 1, 2, 3])
72 >>> c = a−b
73 >>> c
74 array ([20, 29, 38, 47])
```

The `NumPy` module

```
75 >>> b**2
76 array([0, 1, 4, 9])
77 >>> 10*sin(a)
78 array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
79 >>> a<35
80 array([True, True, False, False], dtype=bool)
```

▶ Elementwise product vs matrix product:

```
81 >>> A = array( [[1,1],
82 ...             [0,1]] )
83 >>> B = array( [[2,0],
84 ...             [3,4]] )
85 >>> A*B                        # elementwise product
86 array([[2, 0],
87        [0, 4]])
88 >>> dot(A,B)                   # matrix product
89 array([[5, 4],
90        [3, 4]])
```

The NumPy module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ Array indexing and slicing:

```
91  >>> a
92  array ([[ 0,  1,  2,  3,  4],
93           [ 5,  6,  7,  8,  9],
94           [10, 11, 12, 13, 14]])
95  >>> a[0]
96  array([0, 1, 2, 3, 4])
97  >>> a[1]
98  array([5, 6, 7, 8, 9])
99  >>> a[0:2]
100 array([[0, 1, 2, 3, 4],
101          [5, 6, 7, 8, 9]])
102 >>>
103 >>> a[0, 0]
104 0
105 >>> a[1, 2]
106 7
107 >>> a[0:2, 0:2]
108 array([[0, 1],
109          [5, 6]])
110 >>>
111 >>> a[:,:]
112 array([[ 0,  1,  2,  3,  4],
113          [ 5,  6,  7,  8,  9],
114          [10, 11, 12, 13, 14]])
```

The `NumPy` module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

```
115 >>> a [: , 0]
116 array ([ 0,  5, 10])
117 >>> a [: , 0:1]
118 array ([[ 0],
119        [ 5],
120        [10]])
121 >>>
```

- Iterating over an array:

```
122 >>> for row in a:
123 ...     print row
124 ...
125 [0 1 2 3 4]
126 [5 6 7 8 9]
127 [10 11 12 13 14]
128 >>>
129 >>> for index in xrange(a.shape[1]):
130 ...     print a [: , index ]
131 ...
132 [ 0  5 10]
133 [ 1  6 11]
134 [ 2  7 12]
135 [ 3  8 13]
136 [ 4  9 14]
137 >>>
138 >>> for elt in a.flat:
139 ...     print elt,
140 ...
141 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

The NumPy module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

- Array stacking:

```
142  >>> a = floor(10*random.random((2,2)))
143  >>> a
144  array([[ 1.,  1.],
145         [ 5.,  8.]])
146  >>> b = floor(10*random.random((2,2)))
147  >>> b
148  array([[ 3.,  3.],
149         [ 6.,  0.]])
150  >>> vstack((a,b))
151  array([[ 1.,  1.],
152         [ 5.,  8.],
153         [ 3.,  3.],
154         [ 6.,  0.]])
155  >>> hstack((a,b))
156  array([[ 1.,  1.,  3.,  3.],
157         [ 5.,  8.,  6.,  0.]])
```

The NumPy module

▶ Shallow copying:

```
158 >>> c = a.view()
159 >>> c == a
160 array([[ True,  True,  True,  True,  True],
161        [ True,  True,  True,  True,  True],
162        [ True,  True,  True,  True,  True]], dtype=bool)
163 >>> c is a
164 False
165 >>> a[0,0] = 1000
166 >>> a
167 array([[1000,    1,    2,    3,    4],
168        [   5,    6,    7,    8,    9],
169        [  10,   11,   12,   13,   14]])
170 >>> c
171 array([[1000,    1,    2,    3,    4],
172        [   5,    6,    7,    8,    9],
173        [  10,   11,   12,   13,   14]])
174 >>>
```

The `NumPy` module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ The default copy is the shallow copy:

```
175 >>> a
176 array ([[1000,    1,    2,    3,    4],
177        [   5,    6,    7,    8,    9],
178        [  10,   11,   12,   13,   14]])
179 >>> b = a
180 >>> a[0,0] = 0
181 >>> b
182 array ([[ 0,  1,  2,  3,  4],
183        [ 5,  6,  7,  8,  9],
184        [10, 11, 12, 13, 14]])
```

▶ Deep copying:

```
185 >>> a
186 array ([[ 0,  1,  2,  3,  4],
187        [ 5,  6,  7,  8,  9],
188        [10, 11, 12, 13, 14]])
189 >>> b = a.copy()
190 >>> b[0,0] = 1000
191 >>> a
192 array ([[ 0,  1,  2,  3,  4],
193        [ 5,  6,  7,  8,  9],
194        [10, 11, 12, 13, 14]])
195 >>> b
196 array ([[1000,    1,    2,    3,    4],
197        [   5,    6,    7,    8,    9],
198        [  10,   11,   12,   13,   14]])
```

The NumPy module

▶ Logical arrays:

```
199  >>> a = arange(12).reshape(3,4)
200  >>> b = a > 4
201  >>> b              # b is a boolean with the same shape as a
202  array([[False, False, False, False],
203         [False, True, True, True],
204         [True, True, True, True]], dtype=bool)
205  >>> a[b]                                       # 1d array with the
            selected elements
206  array([ 5,  6,  7,  8,  9, 10, 11])
207  >>>
208  >>> a[b] = 0                                   # All elements of 'a'
         higher than 4 become 0
209  >>> a
210  array([[0, 1, 2, 3],
211         [4, 0, 0, 0],
212         [0, 0, 0, 0]])
```

The `NumPy` module

▶ Simple linear algebra:

```
213 >>> from numpy import *
214 >>> from numpy.linalg import *
215
216 >>> a = array([[1.0, 2.0], [3.0, 4.0]])
217 >>> print a
218 [[ 1.   2.]
219  [ 3.   4.]]
220
221 >>> a.transpose()
222 array([[ 1.,   3.],
223        [ 2.,   4.]])
224
225 >>> inv(a)
226 array([[-2. ,   1. ],
227        [ 1.5,  -0.5]])
228
229 >>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"
230 >>> u
231 array([[ 1.,   0.],
232        [ 0.,   1.]])
233 >>> j = array([[0.0, -1.0], [1.0, 0.0]])
234
235 >>> dot(j, j) # matrix product
236 array([[-1.,   0.],
237        [ 0.,  -1.]])
```

The NumPy module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

```
238  >>> trace(u)  # trace
239  2.0
240
241  >>> y = array([[5.], [7.]])
242  >>> solve(a, y)
243  array([[-3.],
244         [ 4.]])
245
246  >>> eig(j)
247  (array([ 0.+1.j,  0.-1.j]),
248  array([[ 0.70710678+0.j,     0.70710678+0.j],
249         [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]]))
250  Parameters:
251      square matrix
252
253  Returns
254      The eigenvalues, each repeated according to its multiplicity.
255
256      The normalized (unit "length") eigenvectors, such that the
257      column ''v[:,i]'' is the eigenvector corresponding to the
258      eigenvalue ''w[i]''.
```

The NumPy module

▶ Matrices:

```
259 >>> A = matrix ('1.0 2.0; 3.0 4.0')
260 >>> A
261 [[ 1.   2.]
262  [ 3.   4.]]
263 >>> type(A)  # file where class is defined
264 <class 'numpy.matrixlib.defmatrix.matrix'>
265
266 >>> A.T  # transpose
267 [[ 1.   3.]
268  [ 2.   4.]]
269
270 >>> X = matrix ('5.0 7.0')
271 >>> Y = X.T
272 >>> Y
273 [[5.]
274  [7.]]
275
276 >>> print A*Y  # matrix multiplication
277 [[19.]
278  [43.]]
279
280 >>> print A.I  # inverse
281 [[-2.    1. ]
282  [ 1.5 -0.5]]
283
284 >>> solve(A, Y)  # solving linear equation
285 matrix ([[-3.],
286         [ 4.]])
```

The NumPy module

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ Caution: indexing and slicing are different between
matrices and arrays.

```
287 >>> A = arange(12).reshape(3,4)
288 >>> M =   mat(A.copy())
289 >>>
290 >>> print A[:,1]
291 [1 5 9]
292 >>> print M[:,1]
293 [[1]
294  [5]
295  [9]]
296 >>>
```

Outline

Basic elements

The NumPy module

Other useful modules

Other useful modules

▶ SciPy: a module for scientific computing

submodule	contents
cluster	clustering algorithms
constants	physical and mathematical constants
fftpack	fast Fourier transform
integrate	integration and ordinary differential equation solvers
interpolate	interpolation and smoothing splines
io	input and output
linalg	linear algebra
ndimage	N-dimensional image processing
odr	orthogonal distance regression
optimize	optimization and root-finding routines
signal	signal processing
sparse	sparse matrix routines
spatial	spatial data structures and algorithms
special	"special" functions
stats	statistical distributions and functions
weave	integration with C/C++

▶ matplotlib: graphics and plotting

Other useful modules

- ▶ PyCUDA: writing and executing GPU kernels from within Python.
    - ▶ Stay tuned ...

Outline

Basic elements

The NumPy module

Other useful modules

Resources

Introduction to
Python 2 for
statisticians

Will Landau

Basic elements

The NumPy
module

Other useful
modules

▶ Guides:

1. David M. Beazley. *Python Essential Reference: Fourth Edition*. Addison-Wesley, 2009.
2. Tentative NumPy Tutorial. `http://www.scipy.org/Tentative_NumPy_Tutorial`.
3. SciPy Tutorial. `http://docs.scipy.org/doc/scipy/reference/tutorial/general.html`
4. Matplotlib. `http://matplotlib.org/`.

▶ Code from today:

▶ IntroPython.py

That's all for today.

▶ Series materials are available at
  http://will-landau.com/gpu.