Python Introduction

Kasra Hosseini

3rd Munich Earth Skience School
2013-03-11

Monday

Morning (8:00-11:00):

- Python data types
- Flow control
- File I/O
- Functions
- Modules

Afternoon (15:00-18:00):

- Plotting
- NumPy
- Scipy
- Basemap
- Other ways of running Python commands/scripts
- More examples

Outline

- This course will **not** teach you basic programming
- Assume you already know:
  - variables
  - loops
  - conditionals (if / else)
  - standard data types, int, float, string, lists / arrays
  - reading/writing data from files
- We will:
  - show you how to use these in Python
  - present some important concepts when using numpy arrays
  - present a few modules in numpy and scipy
  - give a few examples on how to plot graphs and maps

A few reasons for using Python for Research

Python is an interpreted programming language (i.e. it does not compile!)

1. Free
2. Cross-platform
3. Widely used
4. Well documented
5. Readability
6. Batteries included (Extensive standard libraries)
7. Speed

"Batteries included"

- Extensive standard libraries: (http://docs.python.org/2/library/)
  - Data Compression and Archiving
  - Cryptographic Services
  - Internet Protocols
  - Internet Data Handling
  - Structured Markup Processing Tools
  - Multimedia Services
  - Internationalization
  - Development Tools
  - Multithreading & Multiprocessing
  - Regular expressions
  - Graphical User Interfaces with Tk
  - ...

Python Data Types: Numbers

```
>>> a = 17
>>> type(a)
<type 'int'>
```

Python Data Types: Numbers

```
>>> a = 17
>>> type(a)
<type 'int'>
```

```
>>> b = 17.
>>> type(b)
<type 'float'>
```

Python Data Types: Numbers

```
>>> a = 17
>>> type(a)
<type 'int'>
```

```
>>> b = 17.
>>> type(b)
<type 'float'>
```

```
>>> c=3.0+4.0j
>>> type(c)
<type 'complex'>
```

Python Data Types: Numbers

```
>>> a = 17
>>> a / 10
1
>>> a % 10
7
>>> a / 10.0
1.7
```

Python Data Types: Numbers

```
>>> a = 17
>>> a / 10
1
>>> a % 10
7
>>> a / 10.0
1.7
```

```
>>> int(10.56)
10
>>> float(a)
17.0
```

Python Data Types: Numbers

```
>>> a = 17
>>> a / 10
1
>>> a % 10
7
>>> a / 10.0
1.7
```

```
>>> int(10.56)
10
>>> float(a)
17.0
```

```
>>> c=3.0+4.0j
>>> c.real
3.0
>>> c.imag
4.0
>>> abs(c)   # sqrt(a.real**2 + a.imag**2)
5.0
```

Python Data Types: Numbers

```
>>> a = 17
>>> a = a + 1
>>> a
18
```

Python Data Types: Numbers

```
>>> a = 17
>>> a = a + 1
>>> a
18
```

```
>>> a+=2     # equivalent: a = a + 2
>>> a
20
```

Exercise 1

Python Data Types: Strings

```
>>> 'spam eggs'
'spam eggs'
```

Python Data Types: Strings

```
>>> 'spam eggs'
'spam eggs'
```

```
>>> print """
... Usage: thingy [OPTIONS]
...      -h                Display this message
...      -H hostname       Hostname to connect to
... """
```

Python Data Types: Strings

```
>>> 'spam eggs'
'spam eggs'
```

```
>>> print """
... Usage: thingy [OPTIONS]
...      −h                Display this message
...      −H hostname       Hostname to connect to
... """
```

```
>>> ’sp’ + ’am’
’spam’
>>> ’spam’ ⋆ 10
’spamspamspamspamspamspamspamspamspamspam’
```

Python Data Types: Strings

```
>>> a = "MESS2013 workshop"
>>> a[0]
'M'
>>> a[0:8]
'MESS2013'
>>> a[0:1]
'M' # different than in other languages!
>>> a[-1]
'p'
>>> a[9:] #equivalent a[-8:]
'workshop'
```

Python Data Types: Strings

```
>>> a = "MESS2013 workshop"
>>> a[0]
'M'
>>> a[0:8]
'MESS2013'
>>> a[0:1]
'M' # different than in other languages!
>>> a[-1]
'p'
>>> a[9:] #equivalent a[-8:]
'workshop'
```

```
>>> len(a)
17
```

Python Data Types: Strings

Strings are objects with many useful methods:

```
>>> a = "MESS2013"
>>> a.find('20')
4
```

Python Data Types: Strings

Strings are objects with many useful methods:

```
>>> a = "MESS2013"
>>> a.find('20')
4
```

```
>>> a.lower()
'mess2013'
```

Python Data Types: Strings

Strings are objects with many useful methods:

```
>>> a = "MESS2013"
>>> a.find('20')
4
```

```
>>> a.lower()
'mess2013'
```

```
>>> a.capitalize()
'Mess2013'
```

There are more useful `string` methods like `startswith`, `endswith`, `split`, `join`, `ljust`, `rjust`, `center`, `. . . .` See Python Library Reference.

Exercise 2

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-1]
1234
>>> a[-2]
100
>>> 2*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-1]
1234
>>> a[-2]
100
>>> 2*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

```
>>> a[2] = a[2] + 23 # lists are mutable
>>> a
['spam', 'eggs', 123, 1234]
```

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

```
>>> len(a)
4
```

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

```
>>> len(a)
4
```

```
>>> a[0:2] = [1, 12] # Replace some items
>>> a
[1, 12, 123, 1234]
>>> sum(a) # some over all items
1370
```

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

```
>>> len(a)
4
```

```
>>> a[0:2] = [1, 12] # Replace some items
>>> a
[1, 12, 123, 1234]
>>> sum(a) # some over all items
1370
```

```
>>> a[0:2] = [] # Remove some
>>> a
[123, 1234]
```

Python Data Types: Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

```
>>> len(a)
4
```

```
>>> a[0:2] = [1, 12] # Replace some items
>>> a
[1, 12, 123, 1234]
>>> sum(a) # some over all items
1370
```

```
>>> a[0:2] = [] # Remove some
>>> a
[123, 1234]
```

```
>>> a[1:1] = ['bletch', 'xyzzy'] # Insert some
>>> a
[123, 'bletch', 'xyzzy', 1234]
```

Exercise 3

Python Data Types: Tuples, Boolean & None

Tuples

- Immutable lists created by **round** parantheses
- Parantheses can be ommited in many cases.

```
>>> t = (12345, 54321, 'hello!')
>>> t[0]
12345
```

Python Data Types: Tuples, Boolean & None

Tuples

- Immutable lists created by **round** parantheses
- Parantheses can be ommited in many cases.

```
>>> t = (12345, 54321, 'hello!')
>>> t[0]
12345
```

Boolean

```
>>> type(True)
<type 'bool'>
```

Python Data Types: Tuples, Boolean & None

Tuples

- Immutable lists created by **round** parantheses
- Parantheses can be ommited in many cases.

```
>>> t = (12345, 54321, 'hello!')
>>> t[0]
12345
```

Boolean

```
>>> type(True)
<type 'bool'>
```

None

```
>>> a = None
>>> type(a)
<type 'NoneType'>
```

Python Data Types: Dictionaries

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> print tel
{'sape': 4139, 'jack': 4098}
```

Python Data Types: Dictionaries

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> print tel
{'sape': 4139, 'jack': 4098}
```

```
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Python Data Types: Dictionaries

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> print tel
{'sape': 4139, 'jack': 4098}
```

```
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

```
>>> tel['jack']
4098
>>> del tel['sape']
```

Python Data Types: Dictionaries

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> print tel
{'sape': 4139, 'jack': 4098}
```

```
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

```
>>> tel['jack']
4098
>>> del tel['sape']
```

```
>>> tel.keys()
['guido', 'jack']
>>> 'guido' in tel
True
```

Exercise 4

Flow Control: if-statement

```
if condition-1:
  ...
[elif condition-2:
  ...]
[else:
  ...]
```

Flow Control: if-statement

```
if condition-1:
  ...
[elif condition-2:
  ...]
[else:
  ...]
```

```
>>> x = 42
>>> if x < 0:
...      print 'Negative'
... elif x == 0:
...      print 'Zero'
... elif x == 1:
...      print 'Single'
... else:
...      print 'More'
...
More
```

Exercise 5

Flow Control: while-statement

```
while (condition==True):
        ...
```

Flow Control: while-statement

```
while (condition==True):
        ...
```

```
>>> import time
>>> x = 1
>>> while x < 10:
...     print x
...     x += 1
...     time.sleep(1) # wait one second
...
1
2
3
4
5
6
7
8
9
```

Exercise 6

Flow Control: for-statement

```
>>> a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
...
cat 3
window 6
defenestrate 12
```

Flow Control: for-statement

```
>>> a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
...
cat 3
window 6
defenestrate 12
```

```
>>> for i in range(0, 6, 2):
...     print i,
...
0 2 4
```

Flow Control: for-statement

```
>>> a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
...
cat 3
window 6
defenestrate 12
```

```
>>> for i in range(0, 6, 2):
...     print i,
...
0 2 4
```

```
>>> x = []
>>> for i in range(4):
...     x.append(i**2)
...
>>> x
[0, 1, 4, 9]
```

Flow Control: continue & break

The `break` statement breaks out of the smallest enclosing for or while loop.

```
>>> for i in range(0, 100000):
...     if i>50:
...         print i
...         break
...
51
```

Flow Control: continue & break

The `break` statement breaks out of the smallest enclosing for or while loop.

```
>>> for i in range(0, 100000):
...     if i>50:
...         print i
...         break
...
51
```

The `continue` statement continues with the next iteration of the loop.

```
>>> for i in range(0, 100000):
...     if i!=50:
...         continue
...     print i
...
50
```

Exercise 7

File Handling

Use `open(filename, mode)` to open a file. Returns a File Object.

```
fh = open('/path/to/file', 'r')
```

- Some possible modes:
  - r: Open text file for read.
  - w: Open text file for write.
  - a: Open text file for append.
  - rb: Open binary file for read.
  - wb: Open binary file for write.

Use `close()` to close a given File Object.

```
fh.close()
```

Reading Files

Read a quantity of data from a file:

```
s = fh.read( size ) # size: number of bytes to read
```

Reading Files

Read a quantity of data from a file:

```
s = fh.read( size ) # size: number of bytes to read
```

Read entire file

```
s = fh.read()
```

Reading Files

Read a quantity of data from a file:

```
s = fh.read( size ) # size: number of bytes to read
```

Read entire file

```
s = fh.read()
```

Read one line from file:

```
s = fh.readline()
```

Reading Files

Read a quantity of data from a file:

```
s = fh.read( size ) # size: number of bytes to read
```

Read entire file

```
s = fh.read()
```

Read one line from file:

```
s = fh.readline()
```

Get all lines of data from the file into a list:

```
list = fh.readlines()
```

Reading Files

Read a quantity of data from a file:

```
s = fh.read( size ) # size: number of bytes to read
```

Read entire file

```
s = fh.read()
```

Read one line from file:

```
s = fh.readline()
```

Get all lines of data from the file into a list:

```
list = fh.readlines()
```

Iterate over each line in the file:

```
for line in fh:
    print line,
```

Writing Files

Write a string to the file:

```
fh.write( string )
```

Writing Files

Write a string to the file:

```
fh.write( string )
```

Write several strings to the file:

```
fh.writelines( sequence )
```

Exercise 8

Functions

```
def func(args):
      ....
  return
```

Functions

```
def func(args):
      ....
  return
```

```
def birthday(name):
    print "Happy birthday, " + name + "!"
```

Functions

```
def func(args):
      ....
  return
```

```
def birthday(name):
    print "Happy birthday, " + name + "!"
```

```
def birthday(name):
    print "Happy birthday, %s!" %(name)
```

Functions

```
def func(args):
      ....
  return
```

```
def birthday(name):
    print "Happy birthday, " + name + "!"
```

```
def birthday(name):
    print "Happy birthday, %s!" %(name)
```

```
>>> birthday("Katherine")
Happy birthday, Katherine!
```

Functions

```
>>> print birthday("Katherine")
Happy birthday, Katherine!
None
```

Functions

```
>>> print birthday("Katherine")
Happy birthday, Katherine!
None
```

```
def birthday(name):
    return "Happy birthday, %s!" %(name)

>>> print birthday("Katherine")
Happy birthday, Katherine!
```

Functions

```
>>> print birthday("Katherine")
Happy birthday, Katherine!
None
```

```
def birthday(name):
    return "Happy birthday, %s!" %(name)

>>> print birthday("Katherine")
Happy birthday, Katherine!
```

```
def birthday(name='Kasra'):
    return "Happy birthday, %s!" %(name)

>>> print birthday()
Happy birthday, Kasra!

>>> print birthday("Katherine")
Happy birthday, Katherine!
```

Exercise 9

Modules

Importing functionality of a module the normal and safe way:

```
>>> import math
```

Modules

Importing functionality of a module the normal and safe way:

```
>>> import math
```

```
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

Modules

Importing functionality of a module the normal and safe way:

```
>>> import math
```

```
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

Importing directly into the local namespace:

```
>>> from math import *
>>> pi
3.141592653589793
>>> cos(pi)
-1.0
```

Modules

Import module under a different/shorter name:

```
>>> import math as m
>>> m.cos(m.pi)
-1.0
```

Import only what is needed:

```
>>> from math import pi, cos
>>> cos(pi)
-1.0
```

Exercise 10

Plotting

Matplotlib is *the* plotting library for Python.

- syntax is close to Matlab's plotting commands
- advanced users can control all details of the plots

We need to import `matplotlib` for the following examples:

```
>>> import matplotlib.pyplot as plt
>>> x = [0, 2, 2.5]
>>> plt.plot(x)
[<matplotlib.lines.Line2D object at 0x3372e10>]
>>> plt.show()
```

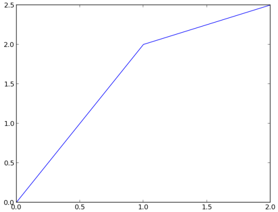
Plotting

Matplotlib is *the* plotting library for Python.

- syntax is close to Matlab's plotting commands
- advanced users can control all details of the plots

We need to import `matplotlib` for the following examples:

```
>>> import matplotlib.pyplot as plt
>>> x = [0, 2, 2.5]
>>> plt.plot(x)
[<matplotlib.lines.Line2D object at 0x3372e10>]
>>> plt.show()
```

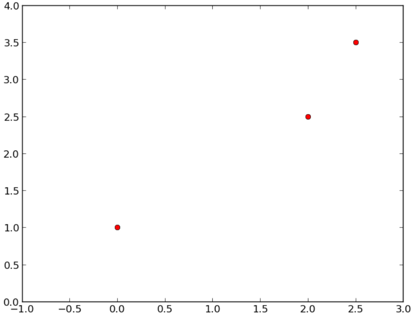


Plotting

```
>>> x = [0, 2, 2.5]
>>> y = [1, 2.5, 3.5]
>>> plt.plot(x, y, 'ro')
>>> plt.xlim(-1, 3)
>>> plt.ylim(0, 4)
>>> plt.show()
```

Plotting

```
>>> x = [0, 2, 2.5]
>>> y = [1, 2.5, 3.5]
>>> plt.plot(x, y, 'ro')
>>> plt.xlim(-1, 3)
>>> plt.ylim(0, 4)
>>> plt.show()
```



Plotting

See the Matplotlib homepage for basic plotting commands and especially the Matplotlib Gallery for many plotting examples with source code!
http://matplotlib.org/
http://matplotlib.org/gallery.html

NumPy

We need to import `numpy` for the following examples:

```
import numpy as np
```

NumPy

We need to import numpy for the following examples:

```
import numpy as np
```

Numpy arrays:

```
>>> a = np.array( [2, 3, 4] )
>>> a
array([2, 3, 4])
>>> type(a)
<type 'numpy.ndarray'>
```

NumPy

We need to import `numpy` for the following examples:

```
import numpy as np
```

Numpy arrays:

```
>>> a = np.array( [2, 3, 4] )
>>> a
array([2, 3, 4])
>>> type(a)
<type 'numpy.ndarray'>
```

```
>>> b = np.array( [ (1.5, 2, 3), (4, 5, 6) ] )
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

NumPy

```
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

NumPy

```
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

```
>>> b.ndim     # number of dimensions
2
>>> b.shape    # the dimensions
(2, 3)
>>> b.dtype    # the type (8 byte floats)
dtype('float64')
```

NumPy

```
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

```
>>> b.ndim     # number of dimensions
2
>>> b.shape    # the dimensions
(2, 3)
>>> b.dtype    # the type (8 byte floats)
dtype('float64')
```

```
>>> c = np.array( [ [1, 2], [3, 4] ],
      dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

NumPy

Create arrays:

```
>>> np.zeros( (3, 4) )  # parameter specify the shape
array([[0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.]])
```

NumPy

Create arrays:

```
>>> np.zeros( (3, 4) )  # parameter specify the shape
array([[0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.]])
```

```
>>> np.ones( (2, 3, 4), dtype=int16 ) # dtype specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
```

Supported data types: bool, uint8, uint16, uint32, uint64, int8, int16, int32, int64, float32, float64, float96, complex64, complex128, complex192

NumPy

```
>>> np.empty( (2,3) )
array([[  3.73603959e-262,   ...,   ...],
       [  5.30498948e-313,   ...,   ...]])
```

NumPy

```
>>> np.empty( (2,3) )
array([[  3.73603959e-262,   ...,   ...],
       [  5.30498948e-313,   ...,   ...]])
```

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
```

NumPy

```
>>> np.empty( (2,3) )
array([[  3.73603959e-262,   ...,   ...],
       [  5.30498948e-313,   ...,   ...]])
```

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
```

```
>>> np.arange( 0, 2, 0.3 ) # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

NumPy

```
>>> np.empty( (2,3) )
array([[  3.73603959e-262,   ...,   ...],
       [  5.30498948e-313,   ...,   ...]])
```

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
```

```
>>> np.arange( 0, 2, 0.3 ) # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

```
>>> np.linspace( 0, 2, 9 ) # 9 numbers from 0 to 2
array([ 0.  ,  0.25,  0.5 ,  0.75, ...,  2.  ])
```

NumPy

```
>>> np.empty( (2,3) )
array([[  3.73603959e-262,   ...,   ...],
       [  5.30498948e-313,   ...,   ...]])
```

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
```

```
>>> np.arange( 0, 2, 0.3 ) # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

```
>>> np.linspace( 0, 2, 9 ) # 9 numbers from 0 to 2
array([ 0.  ,  0.25,  0.5 ,  0.75, ...,  2.  ])
```

```
>>> x = np.linspace( 0, 2*pi, 100 )
>>> f = np.sin(x)
```

NumPy

```
>>> A = np.array( [[1,1], [0,1]] )
>>> B = np.array( [[2,0], [3,4]] )
>>> A*B  # elementwise product
array([[2, 0],
       [0, 4]])
```

NumPy

```
>>> A = np.array( [[1,1], [0,1]] )
>>> B = np.array( [[2,0], [3,4]] )
>>> A*B  # elementwise product
array([[2, 0],
       [0, 4]])
```

```
>>> np.dot(A,B) # matrix product
array([[5, 4],
       [3, 4]])
```

NumPy

```
>>> A = np.array( [[1,1], [0,1]] )
>>> B = np.array( [[2,0], [3,4]] )
>>> A*B  # elementwise product
array([[2, 0],
       [0, 4]])
```

```
>>> np.dot(A,B) # matrix product
array([[5, 4],
       [3, 4]])
```

```
>>> np.mat(A) * np.mat(B) # matrix product
matrix([[5, 4],
        [3, 4]])
```

There are further functions for array creation, conversions, manipulation, querying, ordering, operations, statistics, basic linear algebra. See NumPy documentation.

NumPy

NumPy subpackages

- random: random number generators for various different distributions
- linalg: linear algebra tools
- fft: discrete Fourier transform
- polynomial: efficiently dealing with polynomials

Exercise 11

SciPy

SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension for Python. Scipy subpackages are:

- cluster: Clustering algorithms
- constants: Physical and mathematical constants
- fftpack: Fast Fourier Transform routines
- 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 matrices and associated routines
- spatial: Spatial data structures and algorithms
- special: Special functions
- stats: Statistical distributions and functions
- weave: C/C++ integration

SciPy

```
>>> import scipy as sc
>>> from scipy import integrate
```

SciPy

```
>>> import scipy as sc
>>> from scipy import integrate
```

```
>>> def sinu(x):
        return sc.sin(x)
```

SciPy

```
>>> import scipy as sc
>>> from scipy import integrate
```

```
>>> def sinu(x):
        return sc.sin(x)
```

```
>>> integrate.quad(sinu, 0, 2*sc.pi)
(2.221501482512777e-16, 4.3998892617845996e-14)
```

SciPy

```
>>> import scipy as sc
>>> from scipy import integrate
```

```
>>> def sinu(x):
        return sc.sin(x)
```

```
>>> integrate.quad(sinu, 0, 2*sc.pi)
(2.221501482512777e-16, 4.3998892617845996e-14)
```

or...

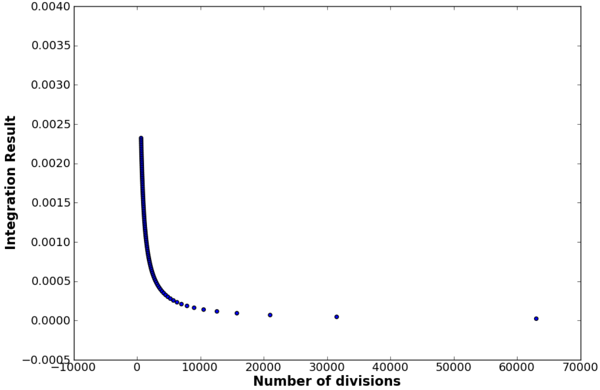
```
>>> integrate.quad(sc.sin, 0, 2*sc.pi)
(2.221501482512777e-16, 4.3998892617845996e-14)
```

SciPy

```
>>> x = sc.arange(0, 2*sc.pi+0.01, 0.01)
>>> integrate.simps(sc.sin(x), dx=0.01)
2.3219645312100389e-05
```

SciPy

```
>>> x = sc.arange(0, 2*sc.pi+0.01, 0.01)
>>> integrate.simps(sc.sin(x), dx=0.01)
2.3219645312100389e-05
```



SciPy

```
>>> import scipy as sc
>>> A = sc.matrix('[1 3 5; 2 5 1; 2 3 8]')
>>> A
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])
```

SciPy

```
>>> import scipy as sc
>>> A = sc.matrix('[1 3 5; 2 5 1; 2 3 8]')
>>> A
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])
```

```
>>> A.I
matrix([[-1.48,  0.36,  0.88],
        [ 0.56,  0.08, -0.36],
        [ 0.16, -0.12,  0.04]])
```

SciPy

```
>>> import scipy as sc
>>> A = sc.matrix('[1 3 5; 2 5 1; 2 3 8]')
>>> A
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])
```

```
>>> A.I
matrix([[-1.48,  0.36,  0.88],
        [ 0.56,  0.08, -0.36],
        [ 0.16, -0.12,  0.04]])
```

```
>>> sc.linalg.inv(A)
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])
```

SciPy

Loading *.mat files generated by Matlab:

```
>> %Matlab
>> mat1 = [1 2 3; 4 5 6; 7 8 9];
>> arr1 = [10 11 12];
>> save test_io.mat mat1 arr1;
```

SciPy

Loading *.mat files generated by Matlab:

```
>> %Matlab
>> mat1 = [1 2 3; 4 5 6; 7 8 9];
>> arr1 = [10 11 12];
>> save test_io.mat mat1 arr1;
```

```
>>> from scipy.io import loadmat
>>> a = loadmat('test_io.mat')
>>> a.keys()
['mat1', '__version__', '__header__', 'arr1', ...]
>>>a['mat1']
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=uint8)
>>> a['arr1']
>>> array([[10, 11, 12]], dtype=uint8)
>>> a = loadmat('test_io.mat',squeeze_me=True)
>>> a['arr1']
array([10, 11, 12], dtype=uint8)
```

SciPy

. . . do the reverse:

```
>>> from scipy.io import savemat
>>> arr2 = a['arr1']
>>> arr2[0] = 20
>>> savemat('test_io_2.mat',
{'mat1':a['mat1'], 'arr2':arr2},oned_as='row')
```

SciPy

. . . do the reverse:

```
>>> from scipy.io import savemat
>>> arr2 = a['arr1']
>>> arr2[0] = 20
>>> savemat('test_io_2.mat',
{'mat1':a['mat1'], 'arr2':arr2},oned_as='row')
```

```
>> load test_io_2.mat
>> mat1
mat1 =
    1    2    3
    4    5    6
    7    8    9
>> arr2
arr2 =
   20   11   12
```

Documentation

- http://docs.scipy.org/doc/
- http://www.scipy.org/Cookbook
- http://scipy-central.org/ (code repository)

Exercise 12

Basemap

- Matplotlib toolkit to plot maps
- Does provide facilities to convert coordinates to one of 25 map projections (using the PROJ library)
- Plotting is done by matplotlib
- Inbuild support for shapefiles

Basemap

A very simple map:

```
>>> from mpl_toolkits.basemap import Basemap
>>> import matplotlib.pyplot as plt
```

Basemap

A very simple map:

```
>>> from mpl_toolkits.basemap import Basemap
>>> import matplotlib.pyplot as plt
```

```
>>> m = Basemap(projection='merc',
llcrnrlat=46.8, urcrnrlat=55.8,
llcrnrlon=4.9, urcrnrlon=16.0, resolution='i')
```

Basemap

A very simple map:

```
>>> from mpl_toolkits.basemap import Basemap
>>> import matplotlib.pyplot as plt
```

```
>>> m = Basemap(projection='merc',
llcrnrlat=46.8, urcrnrlat=55.8,
llcrnrlon=4.9, urcrnrlon=16.0, resolution='i')
```

```
>>> m.drawcountries()
>>> m.drawcoastlines()
```

Basemap

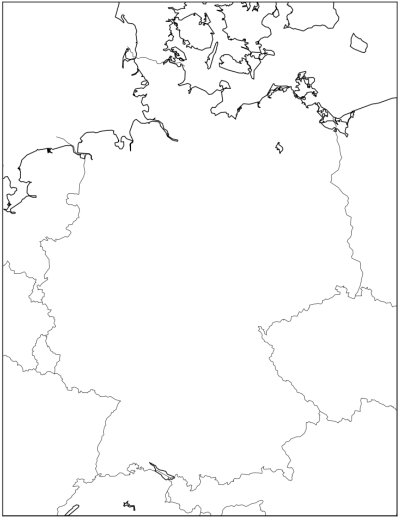
A very simple map:

```
>>> from mpl_toolkits.basemap import Basemap
>>> import matplotlib.pyplot as plt
```

```
>>> m = Basemap(projection='merc',
llcrnrlat=46.8, urcrnrlat=55.8,
llcrnrlon=4.9, urcrnrlon=16.0, resolution='i')
```

```
>>> m.drawcountries()
>>> m.drawcoastlines()
```

```
>>> plt.show()
```



Basemap

…adding a few details:

```
>>> m.drawcountries(linewidth=1.0)
>>> m.drawrivers(color='b')
```

Basemap

. . . adding a few details:

```
>>> m.drawcountries(linewidth=1.0)
>>> m.drawrivers(color='b')
```

```
>>> m.drawmeridians(range(5,16,2),labels=[0,0,0,1])
>>> m.drawparallels(range(47,60),labels=[1,0,0,0])
```

Basemap

...adding a few details:

```
>>> m.drawcountries(linewidth=1.0)
>>> m.drawrivers(color='b')
```

```
>>> m.drawmeridians(range(5,16,2),labels=[0,0,0,1])
>>> m.drawparallels(range(47,60),labels=[1,0,0,0])
```

```
>>> x,y = m(11.567, 48.133)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Munich',va='bottom')
```

Basemap

…adding a few details:

```
>>> m.drawcountries(linewidth=1.0)
>>> m.drawrivers(color='b')
```

```
>>> m.drawmeridians(range(5,16,2),labels=[0,0,0,1])
>>> m.drawparallels(range(47,60),labels=[1,0,0,0])
```

```
>>> x,y = m(11.567, 48.133)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Munich',va='bottom')
```

```
>>> x,y = m(12.036, 47.678)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Berghotel',va='bottom')
```

Basemap

. . . adding a few details:

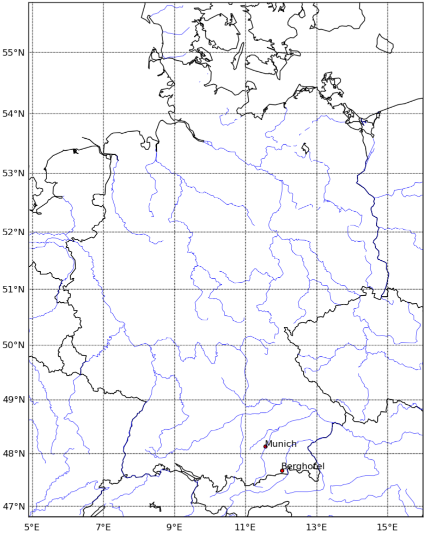
```
>>> m.drawcountries(linewidth=1.0)
>>> m.drawrivers(color='b')
```

```
>>> m.drawmeridians(range(5,16,2),labels=[0,0,0,1])
>>> m.drawparallels(range(47,60),labels=[1,0,0,0])
```

```
>>> x,y = m(11.567, 48.133)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Munich',va='bottom')
```

```
>>> x,y = m(12.036, 47.678)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Berghotel',va='bottom')
```

```
>>> plt.show()
```


55°N
54°N
53°N
52°N
51°N
50°N
49°N
48°N
47°N
5°E
7°E
9°E
11°E
13°E
15°E
Munich
Berghotel

Exercise 13

IPython

- Enhanced interactive Python shell
- Main features
  - Dynamic introspection and help
  - Searching through modules and namespaces
  - Tab completion
  - Complete system shell access
  - Session logging & restoring
  - Verbose and colored exception traceback printouts
  - Highly configurable, programmable (Macros, Aliases)
  - Embeddable

IPython: Getting Help

- Get help for a function:

```
>>> command?
```

- Have a look at the implementation:

```
>>> command??
```

- Search for variables/functions/modules starting with 'ab':

```
>>> ab<Tab>
```

- Which objects are assigned anyway?

```
>>> whos
```

- What attributes/methods are there?

```
>>> object.<Tab>
```

- Get help for a object/class method/attribute:

```
>>> object.command?
```

Modules

Writing your own module called `seismo.py`:

```
"""Some seismological utility functions."""

import math

def lame_parameters(alpha, beta, density):
    """ Convert seismic velocities to Lame's parameters.
        Returns Lame's parameters as (lambda, mu)."""
    return ((alpha ** 2 - 2.0 * beta ** 2) * density,
            beta ** 2 * density)

def velocities(lambd, mu, density):
    """ Convert lame parameters to seismic velocities.
        Returns tuple with velocities (alpha, beta). """
    return (math.sqrt((lambd + 2.0 * mu) / density),
            math.sqrt(mu / density))
```

Modules

Using your module as any other module:

```
>>> import seismo
>>> seismo.lame_parameters(4000., 2100., 2600.)
(18668000000.0, 11466000000.0)
>>> _
(18668000000.0, 11466000000.0)
>>> (_+(2600,))
(18668000000.0, 11466000000.0, 2600)
>>> seismo.velocities(*(_+(2600,)))
(4000.0, 2100.0)
```

Modules

Help!

```
>>> import seismo
>>> help(seismo)
```

```
Help on module seismo:

NAME
    seismo - Some seismological utility functions.

FILE
    /obspy_git/branches/docs/sed_2012/seismo.py

FUNCTIONS
    lame_parameters(alpha, beta, density)
        Convert seismic velocities to Lame's parameters.
        Returns Lame's parameters as (lambda, mu).

    velocities(lambd, mu, density)
        Convert lame parameters to seismic velocities.
        Returns tuple with velocities (alpha, beta).
```

Modules

You can look at the contents of any module

```
>>> import seismo
>>> dir(seismo)
['__builtins__', '__doc__', '__file__',
'__name__', '__package__', 'lame_parameters',
'math', 'velocities']
```

`dir` without argument looks at local namespace

```
...
>>> dir()
['__builtins__', '__doc__',
 '__name__', '__package__', 'seismo']
```

Classes

Classes consist of..

- Attributes: Variables that store information about the class' current state
- Methods: Functions that allow interactions with the class

Some advantages of using classes..

- Classes know how to behave by themselves
- Users do not need to know the details of the class implementation
- Programs using the classes get shorter and far more readable

Classes

Syntax:

- The `class` keyword introduces a class
- To create an instance of the class, use function notation
- The `__init__()` method is invoked when an instance of the class is created
- Class methods receive a reference to the instance as first argument. By convention it is called `self`
- An *instance object* is an entity encapsulating state (data attributes) and behaviour (methods)
- A *class* is the blueprint from which individual objects (*instances*) are created.

Classes

Example:

```
class Rectangle:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def area(self):
        return self.x * self.y
```

```
>>> r = Rectangle(10,20)
>>> r.area()
200
```

Classes

Inheritance

- Motivation: add functionality but reuse existing code
- A derived class has all the attributes and methods from the base class but can add new attributes and methods
- If any new attributes or methods have the same name as an attribute or method in the base class, it is used instead of the base class version.
- The syntax is simply `class DerivedClass(BaseClass): ...`

Classes

Example:

```
class Square(Rectangle):
    def __init__(self,x):
        self.x = x
        self.y = x
```

```
>>> s = Square(5)
>>> s.area()
25
```

Errors and Exceptions

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

Errors and Exceptions

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

```
>>> 4 + muh*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'muh' is not defined
```

Errors and Exceptions

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

```
>>> 4 + muh*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'muh' is not defined
```

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

Errors and Exceptions

Handling Exceptions:

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print "division by zero!"
    except TypeError:
        print "unsupported type!"
    else:
        print "result is", result
```

```
>>> divide(2, 1)
result is 2
>>> divide(2, 0)
division by zero!
>>> divide(2, 'bbb')
unsupported type!
```

Errors and Exceptions

More generic Exception handling:

```
def divide(x, y):
    try:
        result = x / y
    except Exception, e:
        print "Generic exception! ", e
    else:
        print "result is", result
```

```
>>> divide(3.,'blub')
Generic exception!  unsupported operand type(s)
for /: 'float' and 'str'

>>> divide(3.,0)
Generic exception!  float division by zero
```

Credits

- The Python Tutorial (http://docs.python.org/tutorial/)
- Sebastian Heimann - The Informal Python Boot Camp (http://emolch.org/pythonbootcamp.html)
- Software Carpentry (http://software-carpentry.org/4_0/python/)
- Python Scripting for Computational Science, Hans Petter Langtangen
- Matplotlib for Python Developers, Sandro Tosi