ObsPy workshop 2012
Python Introduction

2012-09-06

Program Thursday

Morning: Introduction to Python data types, flow control, file i/o, functions, modules, classes, errors and exceptions, plotting + exercises

Afternoon: Introduction to numpy, scipy, basemap, pyproj and quakepy + exercises

Coffee breaks at 10:30am and 3pm
Lunch break from 12:00pm to 1pm

Wifi
SSID: public
User: obspy
Password: sedworkshop

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

1. Readability
2. Batteries included
3. Speed
4. Language Interoperability

Readability

Guido van Rossum (Python's original author)

> *This emphasis on readability is no accident. As an object-oriented language, Python aims to encourage the creation of reusable code. Even if we all wrote perfect documentation all of the time, code can hardly be considered reusable if it's not readable. Many of Python's features, in addition to its use of indentation, conspire to make Python code highly readable.*

"Batteries included"

- Extensive standard libraries:
  - 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
  - ...

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?
```

Python Data Types: Numbers

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

Python Data Types: Numbers

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

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

Python Data Types: Numbers

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

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

```
>>> _
1.7
>>> type(_)
<type 'float'>
```

Python Data Types: Numbers

```
>>> x = y = z = 0
```

Python Data Types: Numbers

```
>>> x = y = z = 0
```

```
>>> a=3.0+4.0j
>>> float(a)
Traceback (most recent call last):
\dots
TypeError: can't convert complex to float
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)  # 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
>>> a
20
```

Python Data Types: Numbers

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

```
>>> a+=2
>>> a
20
```

```
>>> a++
     a++
        ^
SyntaxError: invalid syntax
```

Python Data Types: Strings

```
>>> 'spam eggs'
'spam eggs'
>>> "doesn't"
"doesn't"
```

Python Data Types: Strings

```
>>> 'spam eggs'
'spam eggs'
>>> "doesn't"
"doesn't"
```

```
>>> 'doesn\'t'
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

Python Data Types: Strings

```
>>> hello = "This is a rather long string\n\
... containing several lines of text.\n\
...     Note that whitespace at the beginning of \
...  the line is significant."
```

Python Data Types: Strings

```
>>> hello = "This is a rather long string\n\
... containing several lines of text.\n\
...     Note that whitespace at the beginning of \
...  the line is significant."
```

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

Python Data Types: Strings

```
>>> 'sp' + 'am'
'spam'
>>> 'spam' * 10
'spamspamspamspamspamspamspamspamspamspam'
```

Python Data Types: Strings

```
>>> 'sp' + 'am'
'spam'
>>> 'spam' * 10
'spamspamspamspamspamspamspamspamspamspam'
```

```
>>> a = "workshop at the SED"
>>> a[0]
'w'
>>> a[0:1]
'w' # different than in other languages!
>>> a[0:8]
'workshop'
>>> a[-3:]
'SED'
```

Python Data Types: Strings

```
>>> a = 'spam'
>>> a[3] = 'n' # strings are immutable
Traceback (most recent call last):
...
TypeError: 'str' object does not support item assignment
```

Python Data Types: Strings

```
>>> a = 'spam'
>>> a[3] = 'n' # strings are immutable
Traceback (most recent call last):
...
TypeError: 'str' object does not support item assignment
```

```
>>> b = a[:-1] + 'n'
>>> b
'span'
```

Python Data Types: Strings

```
>>> a = 'spam'
>>> a[3] = 'n' # strings are immutable
Traceback (most recent call last):
...
TypeError: 'str' object does not support item assignment
```

```
>>> b = a[:-1] + 'n'
>>> b
'span'
```

```
>>> len(b)
4
```

Python Data Types: Strings

Strings are objects with many useful methods:

```
>>> a = "workshop at the SED"
>>> a.find('at')
9
```

Python Data Types: Strings

Strings are objects with many useful methods:

```
>>> a = "workshop at the SED"
>>> a.find('at')
9
```

```
>>> a.split()
['workshop', 'at', 'the', 'SED']
```

Python Data Types: Strings

Strings are objects with many useful methods:

```
>>> a = "workshop at the SED"
>>> a.find('at')
9
```

```
>>> a.split()
['workshop', 'at', 'the', 'SED']
```

```
>>> '*'.join(_)
'workshop*at*the*SED'
```

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

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[-2]
100
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 2*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

Python Data Types: Lists

```
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23 # lists are mutable
>>> a
['spam', 'eggs', 123, 1234]
```

Python Data Types: Lists

```
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23 # lists are mutable
>>> a
['spam', 'eggs', 123, 1234]
```

```
>>> 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]
```

Python Data Types: Lists

```
>>> a[::-1]
[1234, 'xyzzy', 'bletch', 123]
```

Python Data Types: Lists

```
>>> a[::-1]
[1234, 'xyzzy', 'bletch', 123]
```

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

Python Data Types: Lists

```
>>> a[::-1]
[1234, 'xyzzy', 'bletch', 123]
```

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

```
>>> a[:] = [] # Clear the list
>>> a
[]
```

Python Data Types: Lists

```
>>> a[::-1]
[1234, 'xyzzy', 'bletch', 123]
```

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

```
>>> a[:] = [] # Clear the list
>>> a
[]
```

There are more useful `list` methods like `append`, `insert`, `remove`, `sort`, `pop`, `index`, `reverse`, `...`. See Python Library Reference.

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: 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}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Flow Control: if-statement

```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...         print 'Negative'
... elif x == 0:
...         print 'Zero'
... elif x == 1:
...         print 'Single'
... else:
...         print 'More'
...
More
```

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

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

Flow Control: while-statement

```
>>> import time
>>> i = 1
>>> while True:
...     i = i * 1000 # same as: i *= 1000
...     print repr(i)
...     time.sleep(1) # wait one second
...
1000
1000000
1000000000
1000000000000L # <- type conversion occured!
1000000000000000L
# ... continues until memory is exhausted!
```

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
```

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 )
```

The sys module

`sys.argv` returns a list of strings with the pathname of the script as the first entry and the command line arguments as the following entries.

```
import sys
print sys.argv
```

```
In [4]: run tests.py command1 command2
['tests.py', 'command1', 'command2']
```

The sys module

`sys.argv` returns a list of strings with the pathname of the script as the first entry and the command line arguments as the following entries.

```
import sys
print sys.argv
```

```
In [4]: run tests.py command1 command2
['tests.py', 'command1', 'command2']
```

Get information on your platform:

```
>>> sys.platform
'linux2'
```

The sys module

`sys.path` returns the module search path as a list of strings.

```
>>> sys.path
['','/usr/lib/python2.7',
'/usr/local/src/obspy_git/trunk/obspy.core',
 ...]
```

The sys module

`sys.path` returns the module search path as a list of strings.

```
>>> sys.path
['','/usr/lib/python2.7',
'/usr/local/src/obspy_git/trunk/obspy.core',
 ...]
```

If you have written a python module `mymodule.py` that is located in `/my/path/` and you want to load it into another script you could do the following:

```
>>> sys.path.append('/my/path/')
>>> import mymodule
```

Exercises

Functions, modules, classes, exceptions and plotting

Functions

Functions

Defining a function which returns a Fibonacci series up to n.

```
>>> def fib(n):
...     """Return the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)
...         a, b = b, a+b
...     return result
```

Functions

Defining a function which returns a Fibonacci series up to n.

```
>>> def fib(n):
...     """Return the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)
...         a, b = b, a+b
...     return result
```

Now call the function we just defined:

```
>>> fib(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Functions

```
def birthday2(name, age = 1):
    msg = "Happy birthday, %s! You're %d today."
    print  msg % (name, age)
```

Functions

```
def birthday2(name, age = 1):
    msg = "Happy birthday, %s! You're %d today."
    print  msg % (name, age)
```

```
>>> birthday2("Katherine")
Happy birthday, Katherine! You're 1 today.
```

Functions

```
def birthday2(name, age = 1):
    msg = "Happy birthday, %s! You're %d today."
    print  msg % (name, age)
```

```
>>> birthday2("Katherine")
Happy birthday, Katherine! You're 1 today.
```

```
>>> birthday2(age = 12, name = "Katherine")
Happy birthday, Katherine! You're 12 today.
```

Functions

```
def birthday2(name, age = 1):
    msg = "Happy birthday, %s! You're %d today."
    print  msg % (name, age)
```

```
>>> birthday2("Katherine")
Happy birthday, Katherine! You're 1 today.
```

```
>>> birthday2(age = 12, name = "Katherine")
Happy birthday, Katherine! You're 12 today.
```

```
>>> birthday2("Katherine", 14)
Happy birthday, Katherine! You're 14 today.
```

Modules

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

Caution with `from module import *`

```
>>> from os import *
>>> f = open('Makefile','r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: an integer is required

# open() has been mapped to os.open() !!!
```

The builtin open() could still be accessed though:

```
>>> f = __builtins__.open('Makefile','r')
```

Modules

Less typing while avoiding collisions by binding to local names

```
>>> import math
>>> cos = math.cos
>>> pi = math.pi
>>> cos(pi)
-1.0
```

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
```

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']
```

Errors and Exceptions

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
```

Classes

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
```

Plotting

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()
```

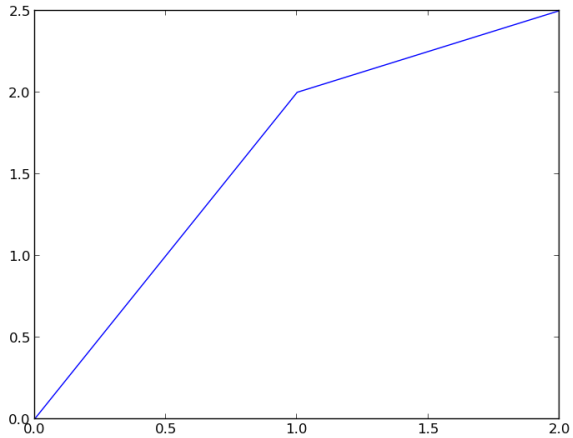
Plotting

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()
```

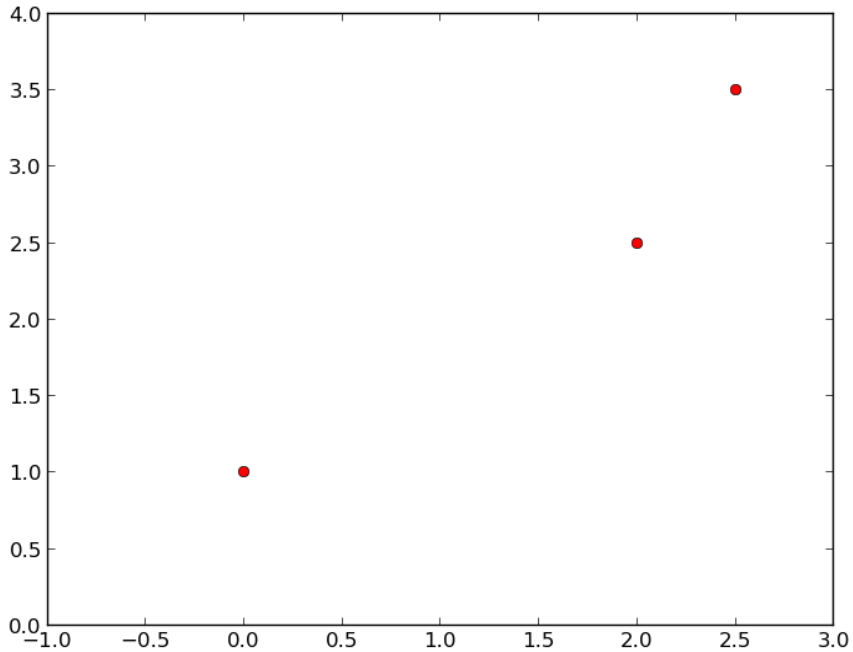


Plotting

```
>>> 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()
```

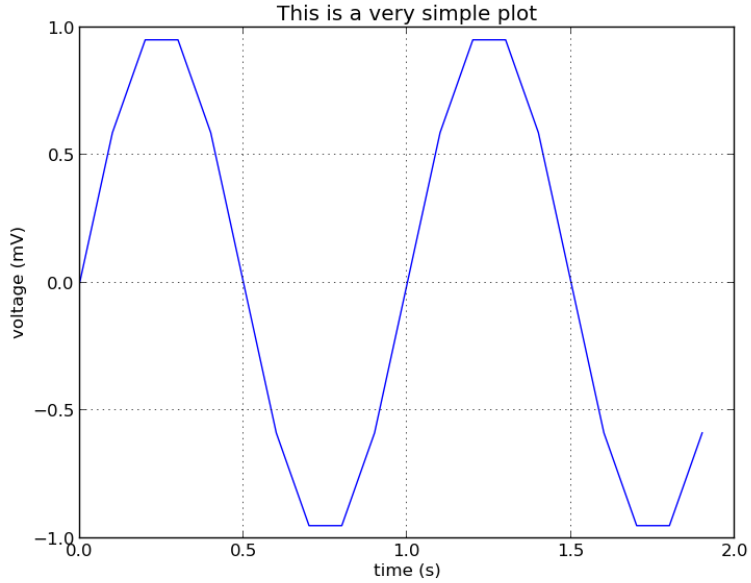


Plotting

```
>>> import math
>>> t = [0.1 * i for i in range(0,20)]
>>> s = [math.sin(2*math.pi*x) for x in t]
>>> plt.plot(t, s, linewidth=1.0)
>>> plt.xlabel('time (s)')
>>> plt.ylabel('voltage (mV)')
>>> plt.title('This is a very simple plot')
>>> plt.grid(True)
>>> plt.show()
```

Plotting

```
>>> import math
>>> t = [0.1 * i for i in range(0,20)]
>>> s = [math.sin(2*math.pi*x) for x in t]
>>> plt.plot(t, s, linewidth=1.0)
>>> plt.xlabel('time (s)')
>>> plt.ylabel('voltage (mV)')
>>> plt.title('This is a very simple plot')
>>> plt.grid(True)
>>> 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.sourceforge.net/index.html
http://matplotlib.sourceforge.net/gallery.html

Exercises

Numpy and Scipy

Numpy

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

NumPy

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

```
>>> c = np.array( [ [1, 2], [3, 4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
>>> d = np.array( [ [1+3j, 2], [3+2.5j, 4] ],
 dtype=complex )
array([[ 1.+3.j ,  2.+0.j ],
       [ 3.+2.5j,  4.+0.j ]])
```

NumPy

You can define your own dtypes. Suppose you have the following array of tuples:

```
>>> x = np.array([[('Christian', 43887651),
('John', 90117628)],
[('Martha', 43887651),
('Stephen', 90117628)]])
>>> x
array([[['Christian', '43887651'],
        ['John', '90117628']],
       [['Martha', '43887651'],
        ['Stephen', '90117628']]],
      dtype='|S9')
>>> dt = np.dtype({'names':('name','phone'),
'formats':('|S8', 'i8')})
>>> x = np.array([[('Christian', 43887651),
('John', 90117628)],
[('Martha', 43887651),
('Stephen', 90117628)]], dtype=dt)
>>> x
array([[('Christia', 43887651), ('John', 90117628)],
       [('Martha', 43887651), ('Stephen', 90117628)]],
      dtype=[('name', '|S8'), ('phone', '<i8')])
```

```
>>> x['name']
array([['Christia', 'John'],
       ['Martha', 'Stephen']],
      dtype='|S8')
>>> x[0,0]
('Christia', 43887651)
>>> x['phone'][0,0]
43887651
>>> x[1] = ('Julia',11324455)
>>> x
array([[('Christia', 43887651), ('John', 90117628)],
       [('Julia', 11324455), ('Julia', 11324455)]],
      dtype=[('name', '|S8'), ('phone', '<i8')])
```

NumPy

A more useful example! Suppose you have a file with entries that look like this:

```
CLC -1.175975e+02 3.581574e+01 soil
SLA -1.172833e+02 3.589095e+01 rock
GRA -1.173662e+02 3.699608e+01 soil
TIN -1.182301e+02 3.705422e+01 soil
```

NumPy

A more useful example! Suppose you have a file with entries that look like this:

```
CLC -1.175975e+02 3.581574e+01 soil
SLA -1.172833e+02 3.589095e+01 rock
GRA -1.173662e+02 3.699608e+01 soil
TIN -1.182301e+02 3.705422e+01 soil
```

```
>>> stdat = np.loadtxt('station_data.txt',
dtype={'names':('stname', 'lon', 'lat', 'substrate'),
'formats': ('|S4', 'f8', 'f8', '|S4')})
>>> stdat
array([('CLC', -117.5975, 35.81574, 'soil'),
       ('SLA', -117.2833, 35.89095, 'rock'),
       ('GRA', -117.3662, 36.99608, 'soil'),
       ('TIN', -118.2301, 37.05422, 'soil')],
      dtype=[('stname', '|S4'), ('lon', '<f8'),
       ('lat', '<f8'), ('substrate', '|S4')])
>>> stdat['stname']
array(['CLC', 'SLA', 'GRA', 'TIN'],
      dtype='|S4')
>>> stdat['lon']
>>> array([-117.5975, -117.2833, -117.3662, -118.2301])
```

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

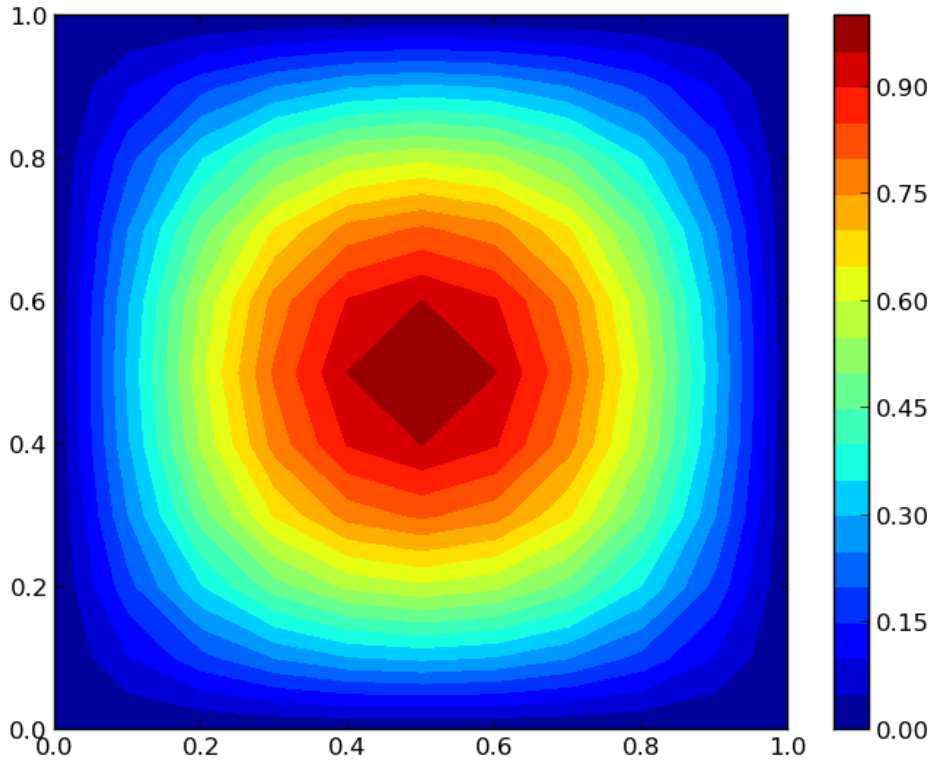
Numpy

Creating grids

```
>>> x = np.arange(0.0,1.1,0.1)
>>> y = x
>>> xx, yy = np.meshgrid(x,y)
>>> xx
array([[ 0. ,  0.1,  0.2, ...,  0.8,  0.9,  1. ],
       [ 0. ,  0.1,  0.2, ...,  0.8,  0.9,  1. ],
       [ 0. ,  0.1,  0.2, ...,  0.8,  0.9,  1. ],
       ...,
       [ 0. ,  0.1,  0.2, ...,  0.8,  0.9,  1. ],
       [ 0. ,  0.1,  0.2, ...,  0.8,  0.9,  1. ],
       [ 0. ,  0.1,  0.2, ...,  0.8,  0.9,  1. ]])
>>> yy
array([[ 0. ,  0. ,  0. , ...,  0. ,  0. ,  0. ],
       [ 0.1,  0.1,  0.1, ...,  0.1,  0.1,  0.1],
       [ 0.2,  0.2,  0.2, ...,  0.2,  0.2,  0.2],
       ...,
       [ 0.8,  0.8,  0.8, ...,  0.8,  0.8,  0.8],
       [ 0.9,  0.9,  0.9, ...,  0.9,  0.9,  0.9],
       [ 1. ,  1. ,  1. , ...,  1. ,  1. ,  1. ]])
```

```
>>> plt.contourf(xx,yy,
np.sin(xx*np.pi)*np.sin(yy*np.pi),20)
>>> plt.colorbar()
>>> plt.show()
```

```
>>> plt.contourf(xx,yy,
np.sin(xx*np.pi)*np.sin(yy*np.pi),20)
>>> plt.colorbar()
>>> plt.show()
```



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

Numpy

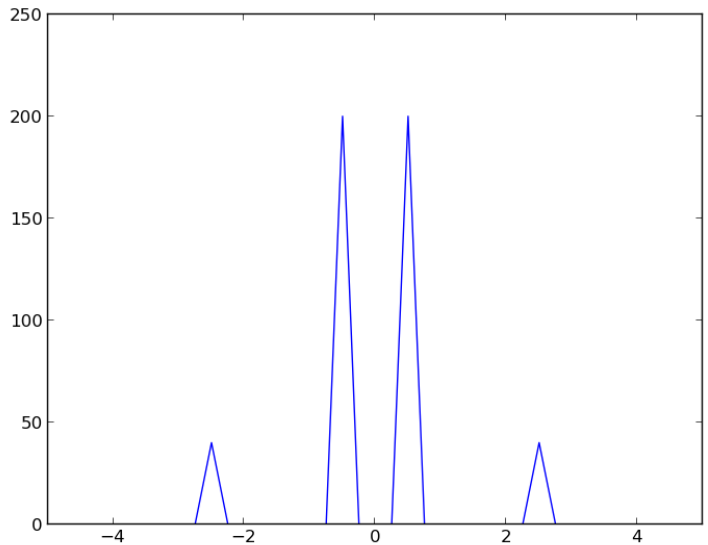
Example fft:

```
>>> sample_rate = 100.0
>>> nsamples = 400
>>> t = np.arange(nsamples) / sample_rate
>>> s = np.cos(2*np.pi*0.5*t)
+ 0.2*np.sin(2*np.pi*2.5*t+0.1)
>>> S = np.fft.fft(s)
>>> freqs = np.fft.fftfreq(nsamples, 1/samp_rate)
>>> plt.plot(freqs,abs(S))
>>> plt.xlim(-5,5)
>>> plt.show()
```

Numpy

Example fft:

```
>>> sample_rate = 100.0
>>> nsamples = 400
>>> t = np.arange(nsamples) / sample_rate
>>> s = np.cos(2*np.pi*0.5*t)
+ 0.2*np.sin(2*np.pi*2.5*t+0.1)
>>> S = np.fft.fft(s)
>>> freqs = np.fft.fftfreq(nsamples, 1/samp_rate)
>>> plt.plot(freqs,abs(S))
>>> plt.xlim(-5,5)
>>> plt.show()
```



Scipy

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

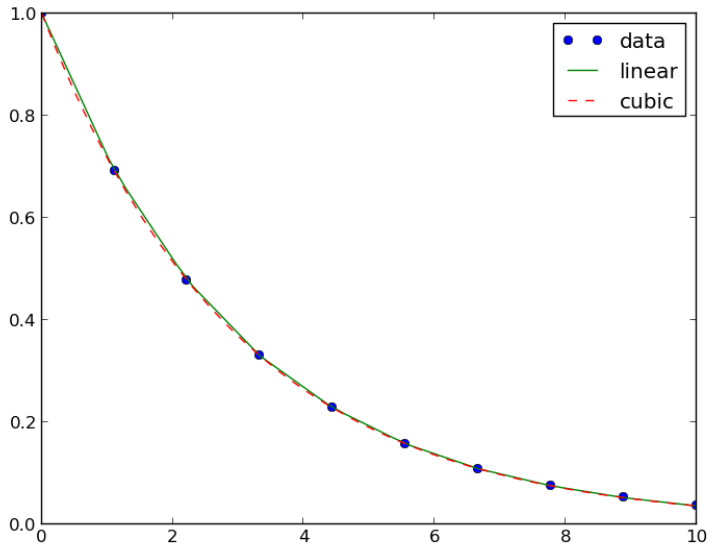
Interpolation

```
>>> from scipy.interpolate import interp1d
>>> x = np.linspace(0, 10, 10)
>>> y = np.exp(-x/3.0)
>>> f = interp1d(x, y)
>>> f2 = interp1d(x, y, kind='cubic')
>>> xnew = np.linspace(0, 10, 40)
>>> plt.plot(x,y,'o')
>>> plt.plot(xnew,f(xnew),'-',xnew, f2(xnew),'--')
>>> plt.legend(['data', 'linear', 'cubic'], loc='best')
>>> plt.show()
```

Scipy

Interpolation

```
>>> from scipy.interpolate import interp1d
>>> x = np.linspace(0, 10, 10)
>>> y = np.exp(-x/3.0)
>>> f = interp1d(x, y)
>>> f2 = interp1d(x, y, kind='cubic')
>>> xnew = np.linspace(0, 10, 40)
>>> plt.plot(x,y,'o')
>>> plt.plot(xnew,f(xnew),'-',xnew, f2(xnew),'--')
>>> plt.legend(['data', 'linear', 'cubic'], loc='best')
>>> plt.show()
```



Scipy

Least-squares data fitting:

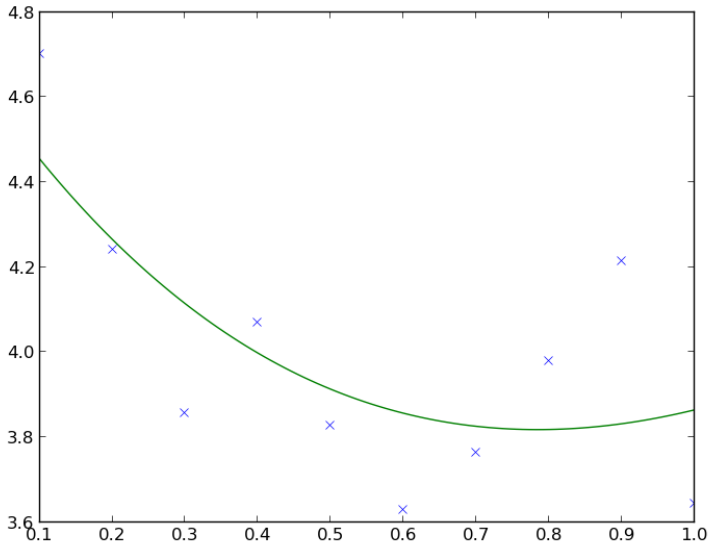
```
>>> from scipy import linalg
>>> c1,c2= 5.0,2.0
>>> i = np.arange(1,11)
>>> xi = 0.1*i
>>> yi = c1*np.exp(-xi)+c2*xi
>>> zi = yi + 0.05*np.max(yi)*np.random.randn(len(yi))
# A*c = zi
>>> A = np.array([np.exp(-xi),xi]).T
>>> c,resid,rank,sigma = linalg.lstsq(A,zi)
>>> c
array([ 5.0525305 ,  2.09272423])
>>> resid
0.70716255651624638
```

Scipy

```
>>> xi2 = np.linspace(0.1,1.0,100)
>>> yi2 = c[0]*np.exp(-xi2) + c[1]*xi2
>>> plt.plot(xi,zi,'x',xi2,yi2)
>>> plt.show()
```

Scipy

```
>>> xi2 = np.linspace(0.1,1.0,100)
>>> yi2 = c[0]*np.exp(-xi2) + c[1]*xi2
>>> plt.plot(xi,zi,'x',xi2,yi2)
>>> plt.show()
```



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
```

Scipy

Reading NetCDF files:

```
>>> # Reading a GMT grid file
>>> from scipy.io import netcdf
>>> nc = netcdf.netcdf_file('ww3.07121615_GLOBAL05.grd')
>>> nc.variables.keys()
>>> ['y', 'x', 'z']
>>> nc.variables['y'].data
>>> array([-58.5, -58. , -57.5, ..., -31. , -30.5, -30. ])
>>> nc.history
>>> '/usr/local/GMT4.5.7/bin/surface
-R146.250000/190.000000/-58.500000/-30.000000
-I0.5 -Gww3.07121615_GLOBAL05.grd
ww3.07121615_GLOBAL05.xyz'
```

Documentation

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

Exercises

Basemap and pyproj

Basemap

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

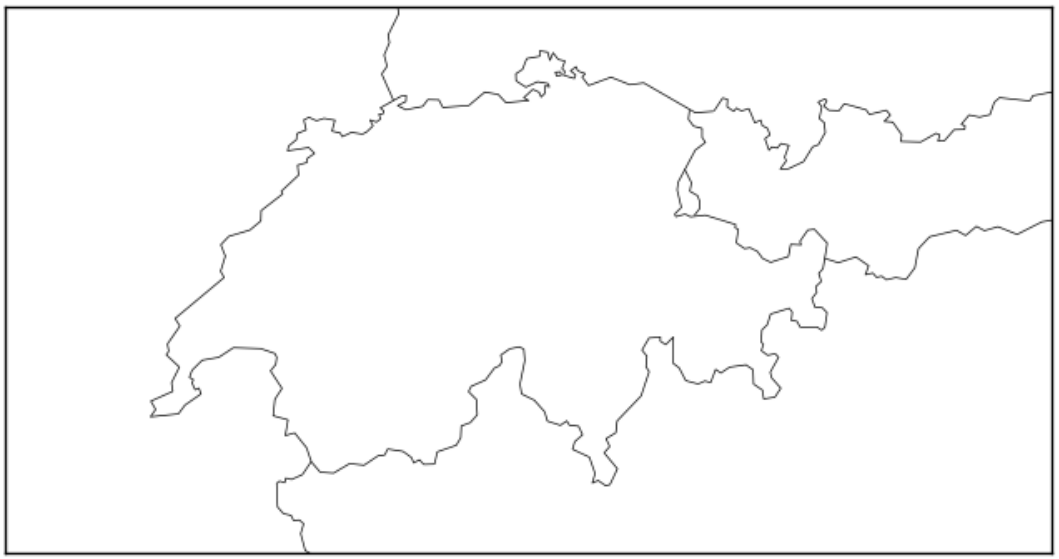
A very simple map:

```
>>> from mpl_toolkits.basemap import Basemap
>>> m = Basemap(projection='merc', llcrnrlat= 45.5,
urcrnrlat=48, llcrnrlon=5, urcrnrlon=12, lat_ts= 47,
resolution='i')
>>> m.drawcountries()
>>> plt.show()
```

Basemap

A very simple map:

```
>>> from mpl_toolkits.basemap import Basemap
>>> m = Basemap(projection='merc', llcrnrlat= 45.5,
urcrnrlat=48, llcrnrlon=5, urcrnrlon=12, lat_ts= 47,
resolution='i')
>>> m.drawcountries()
>>> plt.show()
```



Basemap

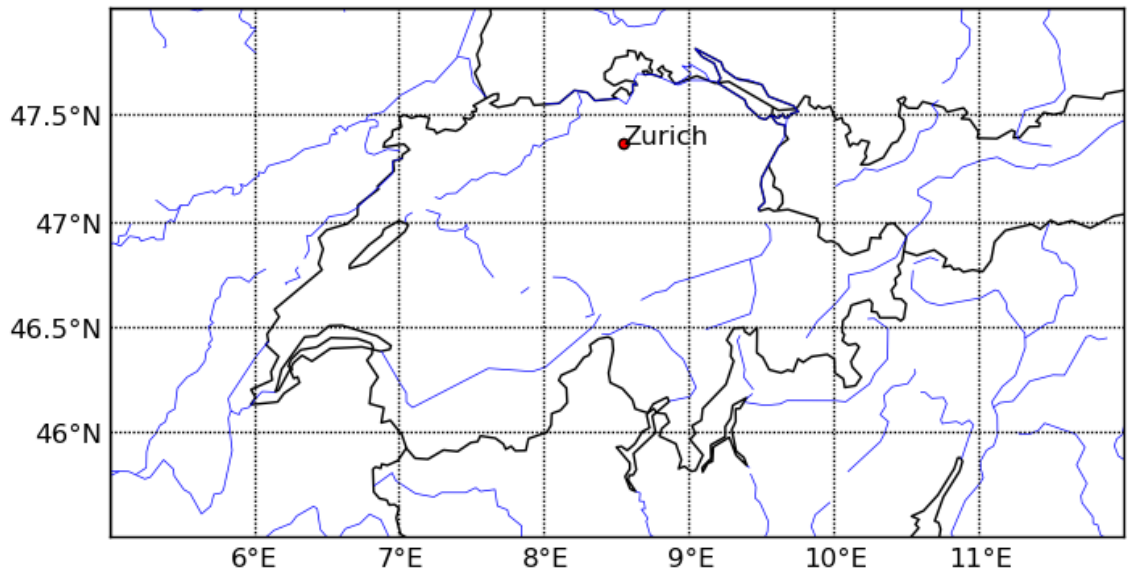
. . . adding a few details:

```
>>> m.drawcoastlines()
>>> m.drawcountries(linewidth=1.0)
>>> m.drawmeridians([6,7,8,9,10,11],labels=[0,0,0,1])
>>> m.drawparallels([46,46.5,47,47.5],labels=[1,0,0,0])
>>> m.drawrivers(color='b')
>>> x,y = m(8.540, 47.372)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Zurich',va='bottom')
>>> plt.show()
```

Basemap

. . . adding a few details:

```
>>> m.drawcoastlines()
>>> m.drawcountries(linewidth=1.0)
>>> m.drawmeridians([6,7,8,9,10,11],labels=[0,0,0,1])
>>> m.drawparallels([46,46.5,47,47.5],labels=[1,0,0,0])
>>> m.drawrivers(color='b')
>>> x,y = m(8.540, 47.372)
>>> m.scatter(x,y,c='r',marker='o')
>>> plt.text(x,y,'Zurich',va='bottom')
>>> plt.show()
```

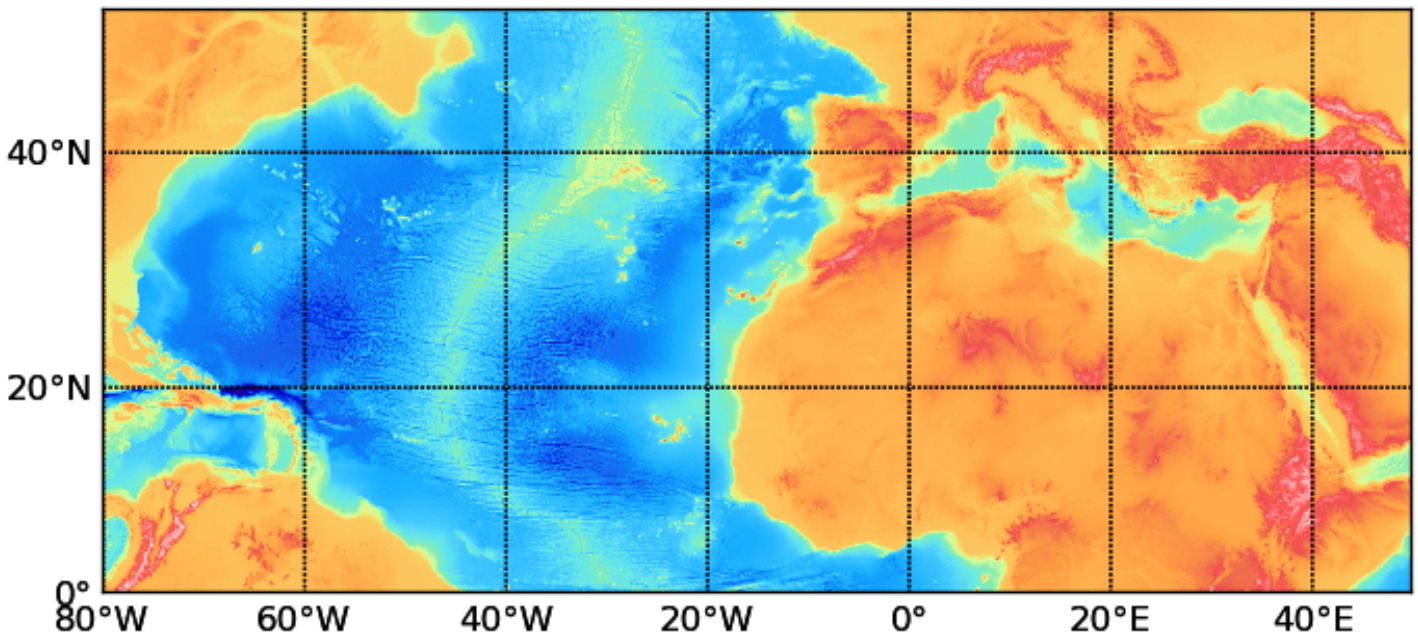


Basemap

Plot topography from the ETOPO1 global relief model:

```
>>> from scipy.io import netcdf
>>> from mpl_toolkits.basemap import Basemap, cm
>>> m = Basemap(projection='merc', llcrnrlat= 0,
urcrnrlat=50, llcrnrlon=-80, urcrnrlon=50, lat_ts= 25,
resolution='i')
>>> nc=netcdf.netcdf_file('ETOPO1_Ice_g_gmt4.grd')
>>> lons = nc.variables['x'].data
>>> lats = nc.variables['y'].data
>>> topoin = nc.variables['z'].data
>>> nx = int((m.xmax-m.xmin)/5000.)+1;
>>> ny = int((m.ymax-m.ymin)/5000.)+1
>>> topodat = m.transform_scalar(topoin,lons,lats,nx,ny)
>>> im = m.imshow(topodat,cm.GMT_haxby)
>>> m.drawparallels(np.arange(0,50,20),labels=[1,0,0,0])
>>> m.drawmeridians(np.arange(-80,50,20),labels=[0,0,0,1])
>>> plt.show()
```

Basemap

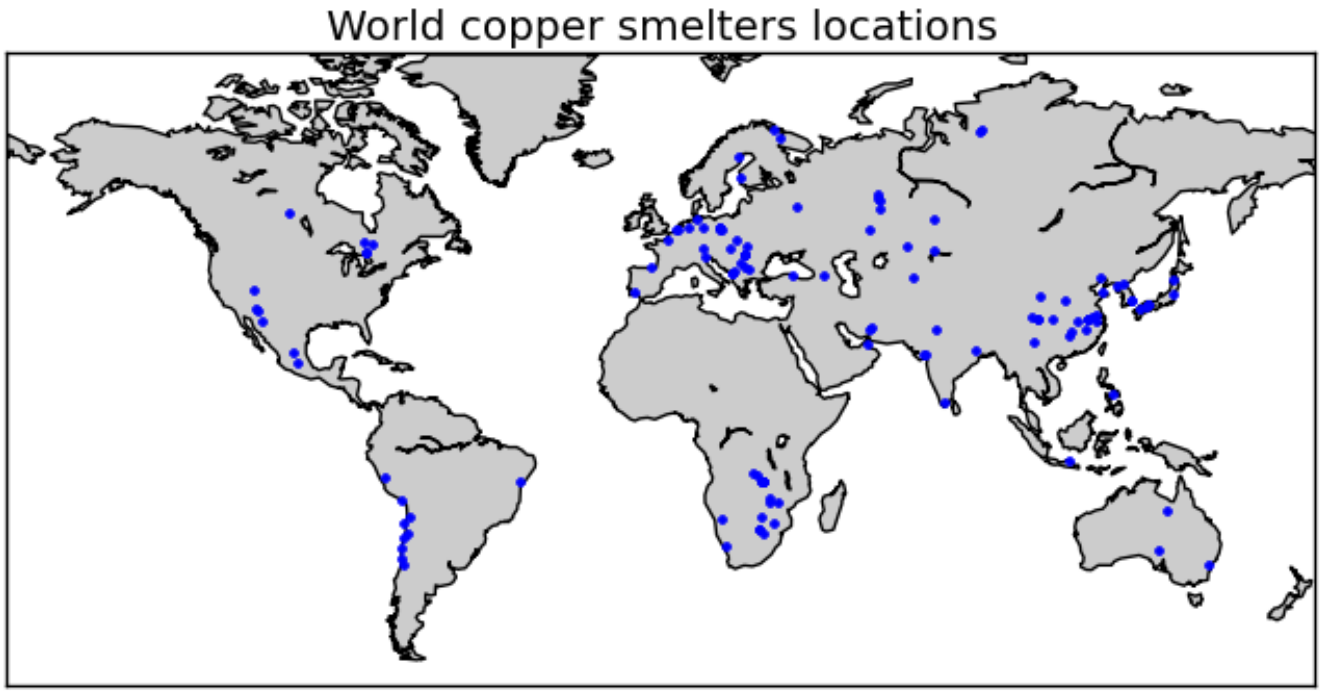


Basemap

Plot information from shapefile data from https://explore.data.gov/d/db3a-9tgn.

```
>>> m = Basemap(projection='mill',
llcrnrlon=-180. ,llcrnrlat=-60,
urcrnrlon=180. ,urcrnrlat=80.)
# assumes 'copper.shp', 'copper.shx' and 'copper.dbf'
# are in the current working directory
>>> s = m.readshapefile('copper', 'copper')
>>> x, y = zip(*m.copper)
>>> m.drawcoastlines()
>>> m.fillcontinents()
>>> plt.plot(x, y, 'b.')
>>> plt.title('World copper smelters locations')
>>> plt.show()
```

Basemap



pyproj

pyproj

- Pyproj provides python bindings to the PROJ4 library.
- Convert from geographic (longitude,latitude) to native map projection (x,y) coordinates and vice versa
- Convert from one map projection coordinate system directly to another
- Perform great circle computations such as determining distance, azimuth and back-azimuth between two given points

Convert longitude and latitude of Zurich into UTM coordinates and back:

```
>>> p = pyproj.Proj(proj="utm",zone=32)
>>> p(8.540, 47.372)
(465271.7360602605, 5246607.042608537)
>>> p(*(_),inverse=True)
(8.54, 47.371999999999986)
```

pyproj

Great circle computation: compute azimuth, back-azimuth and distance between Zurich and Basel and do the inverse.

```
>>> g = pyproj.Geod(ellps='WGS84')
>>> az, baz, dist = g.inv(8.540, 47.372, 7.593, 47.560)
>>> az, baz, dist
(-73.33366215015755, 105.9685077646491, 74392.53849627372)
>>> lon, lat, baz = g.fwd(8.540, 47.372, az, dist)
>>> lon,lat,baz
(7.592999999296916, 47.56000000001562, 105.96850776458967)
```

Exercises

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