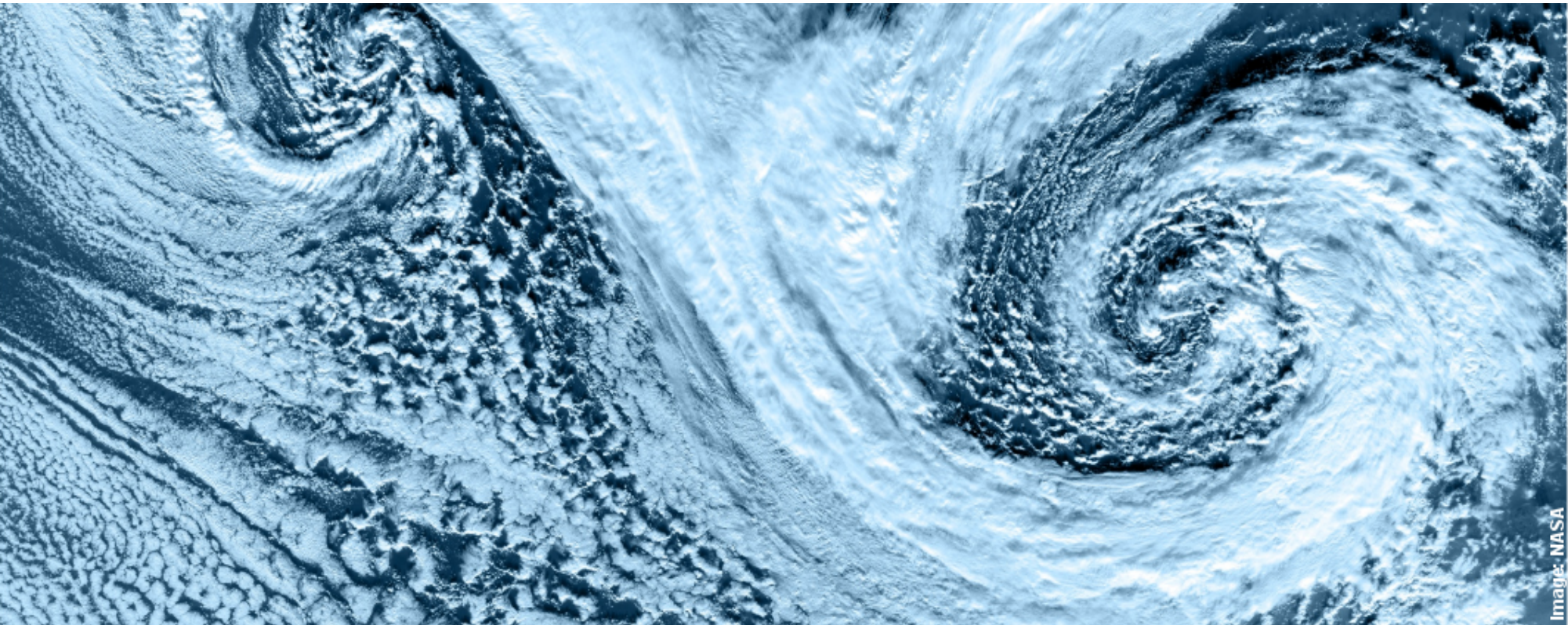
C2SM workshop
Scientific Programming in Python

Harald von Waldow (C2SM/ETHZ)
Nicolas Piaget (Atm.Dyn./ETHZ)
Timm Gross (GIUB/UNIBE)

2015–02–11

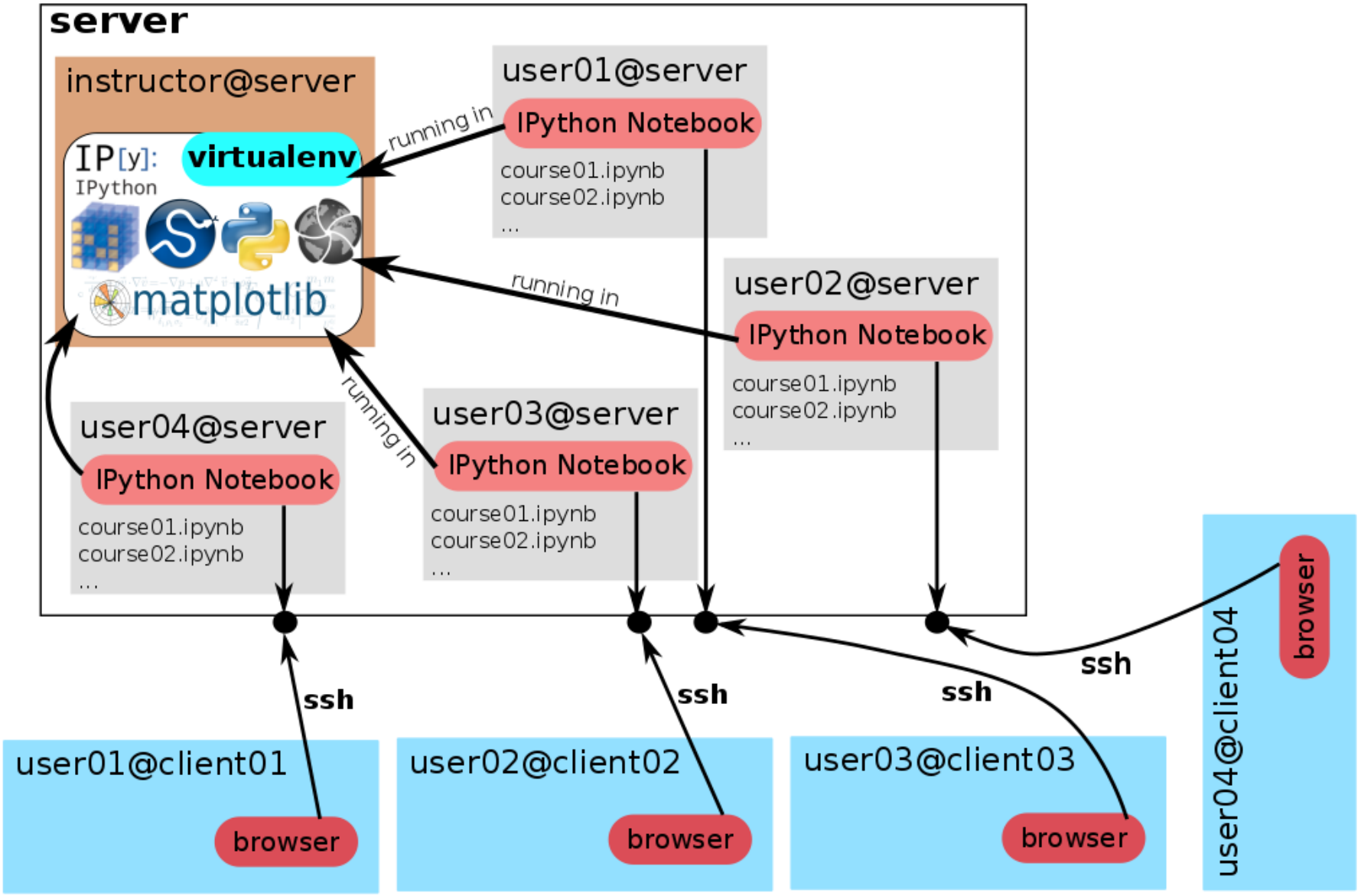


Schedule

09:30	Welcome, organizational matters	Harald
09:45	Python — The big picture	Harald
10:00	Python development tools	Harald
10:15	**Exercise 1**: The IPython Notebook	Harald/Nicolas
10:45	Coffee Break **15 min**	
11:00	**Exercise 2**: Basic Python	Harald
12:00	Lunch Break **1 h**	
13:00	**Exercise 3**: Numpy, Scipy, Matplotlib, netCDF4	Nicolas
14:30	Coffee Break **15 min**	
14:45	**Exercise 4**: More fun with trajectory plots	Nicolas
15:45	Coffee Break **15 min**	
16:00	**Exercise 5**: Easy mapping & projecting	Harald
17:00	Other packages, Outlook	Harald
17:15	Wrap up, Discussion, Questions, Feedback	Harald
17:30?	End	

Organizational Matters

IT-setup



Organizational Matters

IT-setup

- On your local desktop runs a browser.
- The browser connects to a local port (7777), which is forwarded to a port on our server.
- On that server runs the IPython-process, one for everybody, and all data resides there.
- That server is `climcal04.unibe.ch`. You should not need to worry about that.

Organizational Matters

Login

- username: `pywsXX`
- `XX` is `01 ... 17`
- password: `-- censored --`
- After login start a terminal window
- type "./startup"
- the browser should start and load the proper page.

Python — the big picture

History, Versions

- A young and fast moving language by Guido van Rossum:
  - Python 1.0: 1994
  - Python 2.0: 2000
  - Python 2.7: 2010
  - Python 3.0: 2008
  - Python 3.4: 2014
- Python 3.x is incompatible with 2.x.
- New, large projects should be written in 3.x.
- For research code, often depending on exotic modules, use 2.7.
- Some new features can be made available in 2.7 using `from __future__ import XY`

Python — the big picture

History, Versions

- A young and fast moving language by Guido van Rossum:
  - Python 1.0: 1994
  - Python 2.0: 2000
  - Python 2.7: 2010
  - Python 3.0: 2008
  - Python 3.4: 2014
- Python 3.x is incompatible with 2.x.
- New, large projects should be written in 3.x.
- For research code, often depending on exotic modules, use 2.7.
- Some new features can be made available in 2.7 using `from __future__ import XY`

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-oriented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-oriented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Features

- **optimal for fast development**
- documentation well integrated with code
- clean, simple, intuitive
- expressive (= less lines of code)
- scales well with project size

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Features

- **optimal for fast development**
- documentation well integrated with code
- clean, simple, intuitive
- expressive (= less lines of code)
- scales well with project size

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Features

- **optimal for fast development**
- documentation well integrated with code
- clean, simple, intuitive
- expressive (= less lines of code)
- scales well with project size

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Features

- **optimal for fast development**
- documentation well integrated with code
- clean, simple, intuitive
- expressive (= less lines of code)
- scales well with project size

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:
  - object-orented
  - functional programming features

Features

- **optimal for fast development**
- documentation well integrated with code
- clean, simple, intuitive
- expressive (= less lines of code)
- scales well with project size

Python — the big picture

Advantages for scientific computing

- large and fast growing user community
- large and fast growing number of libraries for sci. com.
- well integrated with "bread-and-butter" codes, such as BLAS, LAPACK, Netlib standards such as ODEPACK, ...
- highly extensible, large collection of add-on packages
- easy communication with R, C, Fortran, ...

Python — the big picture

Advantages for scientific computing

- large and fast growing user community
- large and fast growing number of libraries for sci. com.
- well integrated with "bread-and-butter" codes, such as BLAS, LAPACK, Netlib standards such as ODEPACK, ...
- highly extensible, large collection of add-on packages
- easy communication with R, C, Fortran, ...

Python — the big picture

Advantages for scientific computing

- large and fast growing user community
- large and fast growing number of libraries for sci. com.
- well integrated with "bread-and-butter" codes, such as BLAS, LAPACK, Netlib standards such as ODEPACK, ...
- highly extensible, large collection of add-on packages
- easy communication with R, C, Fortran, ...

Python — the big picture

Advantages for scientific computing

- large and fast growing user community
- large and fast growing number of libraries for sci. com.
- well integrated with "bread-and-butter" codes, such as BLAS, LAPACK, Netlib standards such as ODEPACK, ...
- highly extensible, large collection of add-on packages
- easy communication with R, C, Fortran, ...

Python — the big picture

Advantages for scientific computing

- large and fast growing user community
- large and fast growing number of libraries for sci. com.
- well integrated with "bread-and-butter" codes, such as BLAS, LAPACK, Netlib standards such as ODEPACK, ...
- highly extensible, large collection of add-on packages
- easy communication with R, C, Fortran, ...

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, ...

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Other interpreted languages to consider

- R: lingua franca of statistics. There is no choice for advanced stats.
- You better know Python **and** R.
- Matlab: Some specialized toolboxes have (yet) no counterpart in Python.

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Other interpreted languages to consider

- R: lingua franca of statistics. There is no choice for advanced stats.
- You better know Python **and** R.
- Matlab: Some specialized toolboxes have (yet) no counterpart in Python.

Python — the big picture

When to use Python?

ALWAYS!

except:

- You need the speed of C or Fortran
- You can re-use significant code written in other languages.
- You do HPC (using MPI, OpenMP)
- Your can profit from a non-Python tradition in your field or workgroup.
- Something else is better for the task, . . .

Other interpreted languages to consider

- R: lingua franca of statistics. There is no choice for advanced stats.
- You better know Python **and** R.
- Matlab: Some specialized toolboxes have (yet) no counterpart in Python.

Python — the big picture

Philosophy

The Zen of Python (try ‘‘`import this`’’)

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python — the big picture

Philosophy

The Zen of Python (try ‘‘`import this`’’)

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python — the big picture

Philosophy

The Zen of Python (try ‘‘`import this`’’)

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python — the big picture

Philosophy

The Zen of Python (try ‘‘`import this`’’)

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- . . .

Python — the big picture

Philosophy

The Zen of Python (try ‘‘`import this`’’)

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python — the big picture

Philosophy

The Zen of Python (try ‘‘`import this`’’)

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python — the big picture

Philosophy

The Zen of Python (try "`import this`")

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python programmers strive to write *pythonic*

- Don't get lost in beauty
- But if you have trouble understanding your own code ...
- ... after your holidays ...
- "`import this`" and re-factor a bit.

Python — the big picture

Philosophy

The Zen of Python (try "`import this`")

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one– and preferably only one –obvious way to do it.
- ...

Python programmers strive to write *pythonic*

- Don't get lost in beauty
- But if you have trouble understanding your own code ...
- ... after your holidays ...
- "`import this`" and re-factor a bit.

Basic Python

Python shell

```
hvwaldow@mie:~> python
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Typing “python” will call the “raw” Python shell.

Basic Python

Python shell

```
hvwaldow@mie:~> python
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> b = 7
>>> a + b
10
>>>
```

Can be used like a calculator.

Basic Python

Python shell

```
hvwaldow@mie:~> python
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> b = 7
>>> a + b
10
>>> quit()
hvwaldow@mie:~>
```

Call function “quit()” to exit.

Basic Python

IPython shell

```
hvwaldow@mie:~> ipython
WARNING: `-pylab` flag has been deprecated.
    Use `--matplotlib <backend>` and import pylab manually.
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Typing "ipython" will call the IPython shell.

Basic Python

IPython shell

```
hvwaldow@mie:~> ipython
WARNING: `-pylab` flag has been deprecated.
    Use `--matplotlib <backend>` and import pylab manually.
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: x = arange(0, 5*pi, 0.1)

In [2]: y = sin(x) + 0.1*x + rand(len(x))

In [3]:
```

Math functions “magically” loaded.

Basic Python

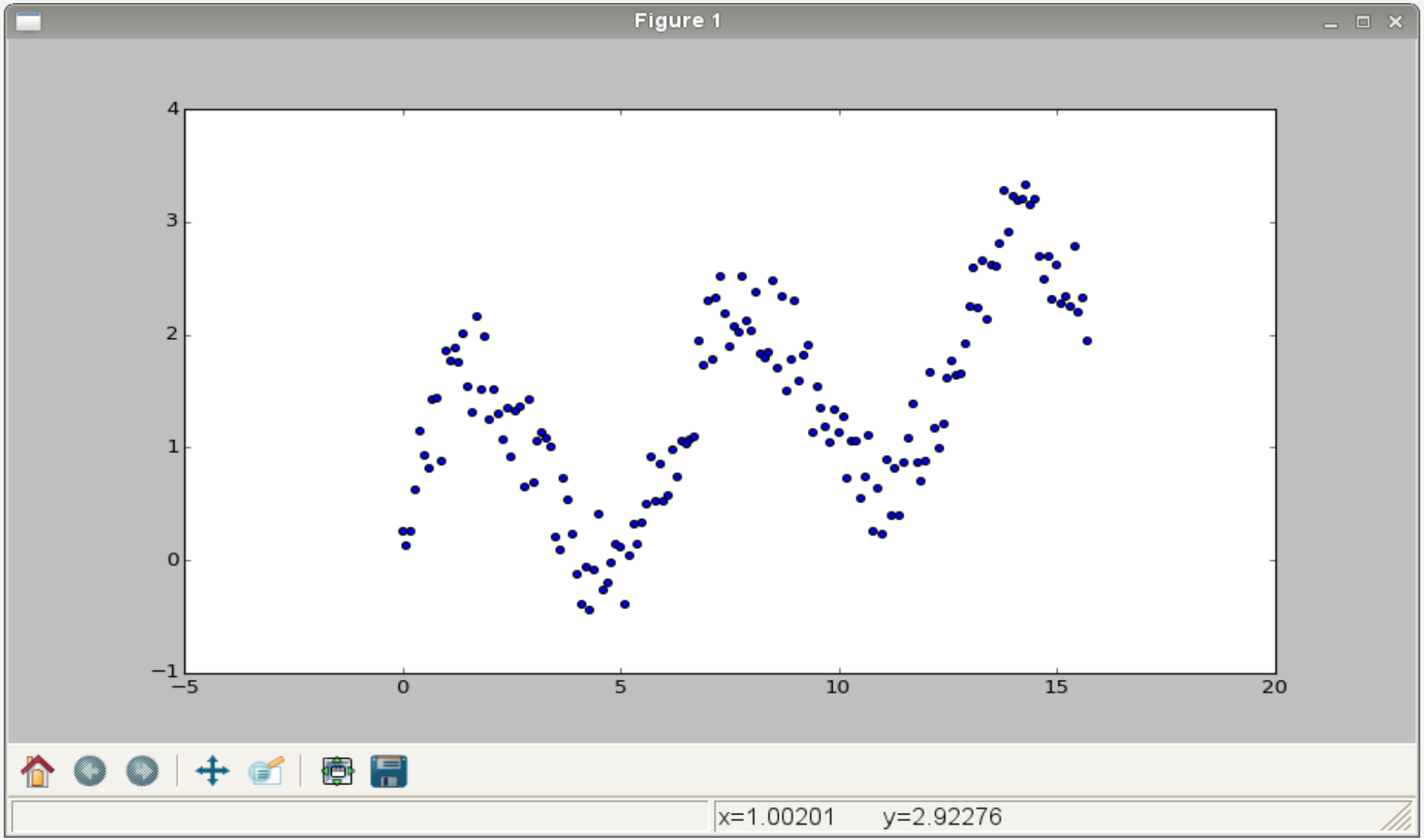
IPython shell

```
hvwaldow@mie:~> ipython
WARNING: `-pylab` flag has been deprecated.
    Use `--matplotlib <backend>` and import pylab manually.
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: x = arange(0, 5*pi, 0.1)

In [2]: y = sin(x) + 0.1*x + rand(len(x))

In [3]: scatter(x,y)
Out[3]: <matplotlib.collections.PathCollection at 0x56bdc50>

In [4]:
```

Plot functions as well.

Basic Python

IPython shell



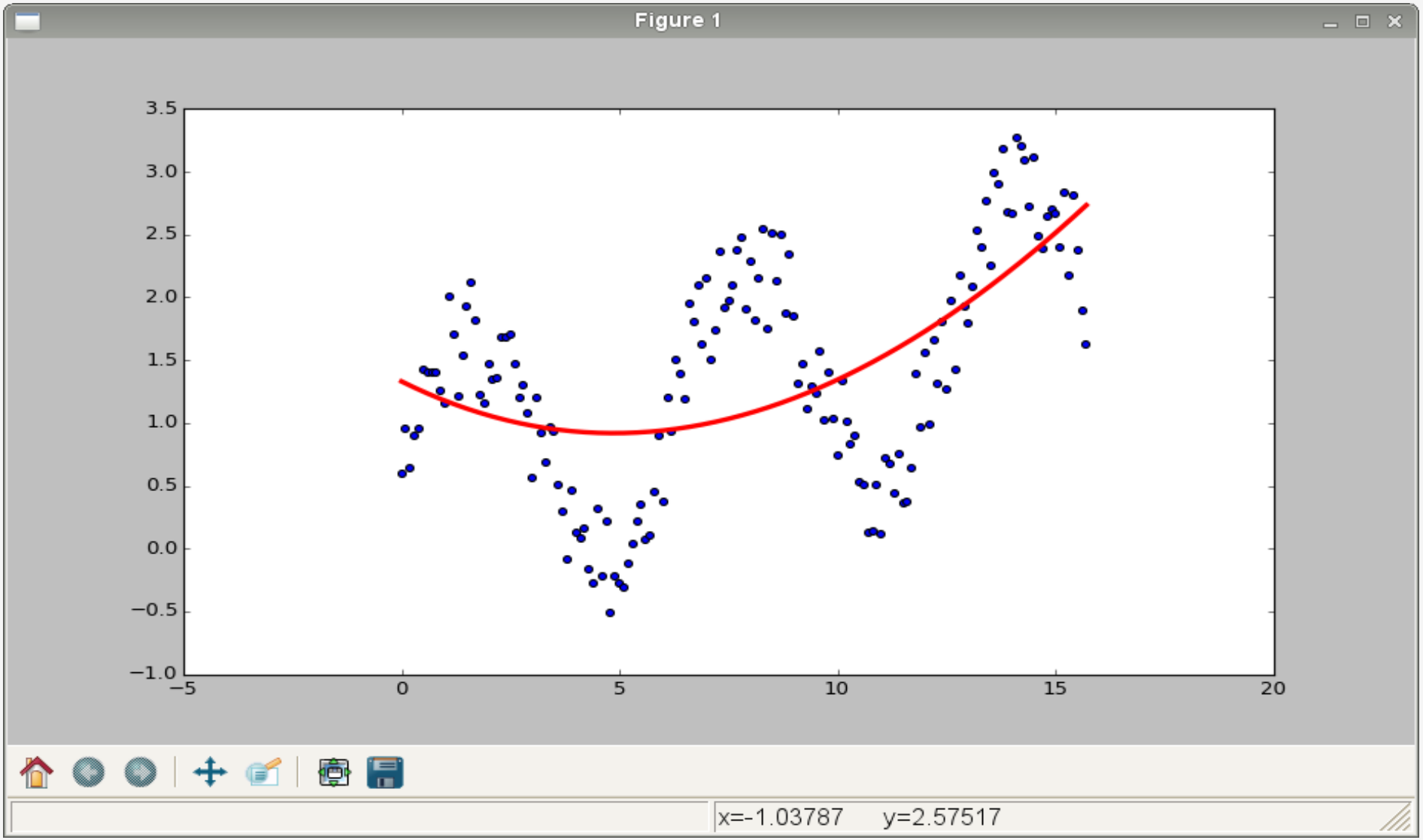
Basic Python

IPython shell

```
Terminal
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:  x = arange(0, 5*pi, 0.1)

In [2]:  y = sin(x) + 0.1*x + rand(len(x))

In [3]: scatter(x,y)
Out[3]: <matplotlib.collections.PathCollection at 0x5944c50>

In [4]: fit = polyfit(x,y,3)

In [5]: plot(x, polyval(fit,x), linewidth=3, c="red")
Out[5]: [<matplotlib.lines.Line2D at 0x5959bd0>]

In [6]:
```

Basic Python

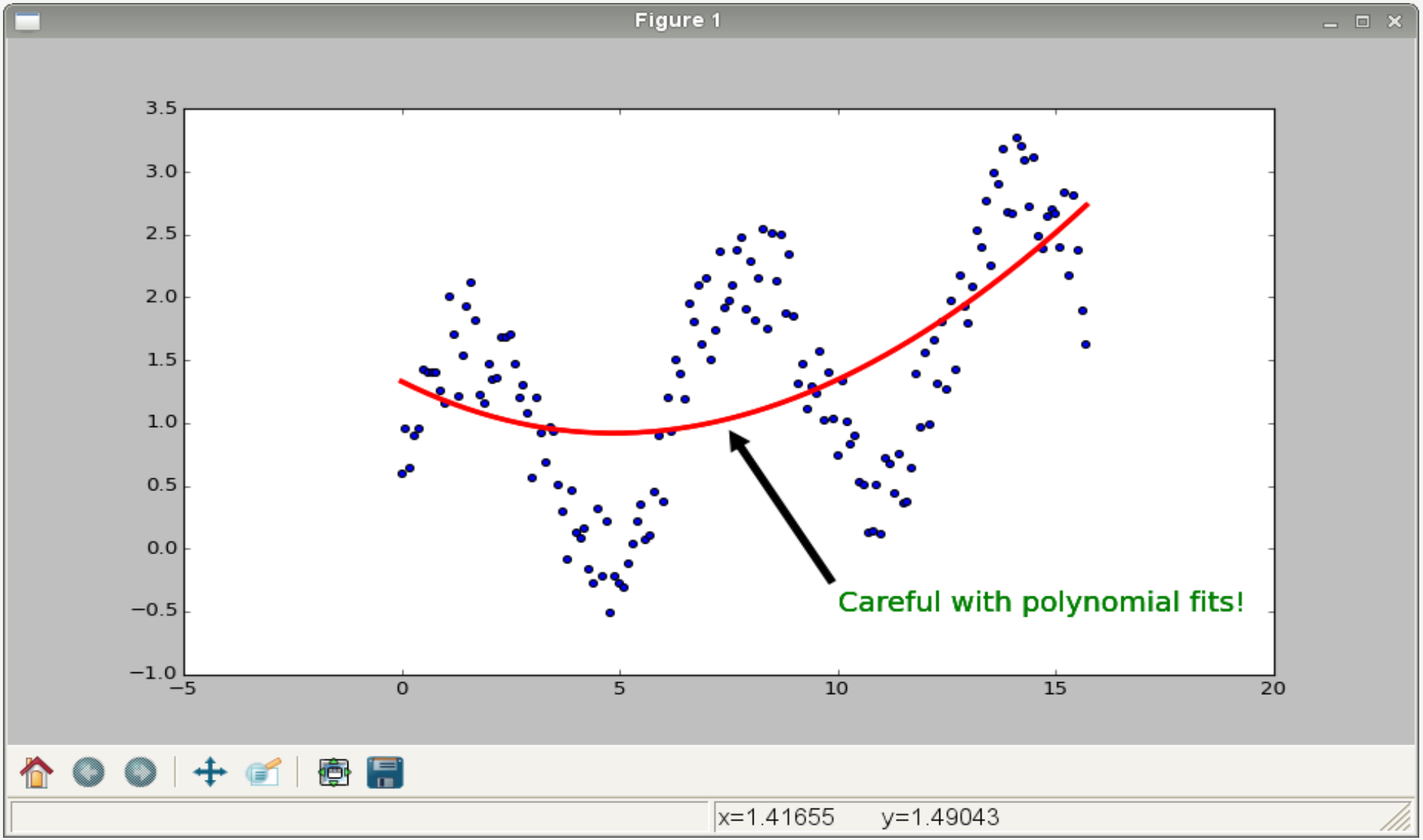
IPython shell



New content can be added interactively.

Basic Python

IPython shell



New content can be added interactively.

Basic Python

Run a python script

emacs@mie.ethz.ch

File Edit Options Buffers Tools Python Help

```
1 ''' Calculate and print Fibonacci-Numbers from from 1 to 20'''
2
3
4 def fib(n):
5     return(n if n < 2 else fib(n-1) + fib(n-2))
6
7 print([fib(n) for n in range(1, 21)])
```

U:---fibonacci.py All (2,0) (Python AC FlyC)

Save file with ending ".py"

Basic Python

Run a python script

```
hvwaldow@mie:~/tmp> python fibonacci.py
```

Call python intepreter with filename as argument.

Basic Python

Run a python script

```
hvwaldow@mie:~/tmp> python fibonacci.py
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
 610, 987, 1597, 2584, 4181, 6765]
hvwaldow@mie:~/tmp>
```

Output goes to stdout.

Basic Python

Run a python script

emacs@mie.ethz.ch

File Edit Options Buffers Tools Python Help

```
1 #!/usr/bin/python
2 ''' Calculate and print Fibonacci-Numbers from from 1 to 20'''
3
4
5 def fib(n):
6     return(n if n < 2 else fib(n-1) + fib(n-2))
7
8 print([fib(n) for n in range(1, 21)])
```

U:---fibonacci.py All (1,17) (Python AC FlyC)

Wrote /home/hvwaldow/tmp/fibonacci.py

"Shebang" syntax works.

Basic Python

Run a python script

```
hvwaldow@mie:~/tmp> chmod u+x fibonacci.py
hvwaldow@mie:~/tmp> ./fibonacci.py
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
 610, 987, 1597, 2584, 4181, 6765]
hvwaldow@mie:~/tmp>
```

Make script executable and call it directly.

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

- A number of sophisticated IDEs available
- Like Matlab, RStudio, Visual Studio, Eclipse ...
- Can be confusing at first
- Many features for program development
- Variable inspection, debugging, ...
- Komodo IDE (commercial)
- Wing IDE (commercial)
- Eclipse/PyDev (free)
- Eric (free)
- Spyder (free)

Python development tools

Python IDEs

Spyder (Python 2.7)

File Edit Search Source Run Interpreters Tools View ?

Editor - /home/hvwaldow/esget/code/esget_fs.py

.temp.py | esget_fs.py*

```python
""" esget_fs implements functions that operate mainly in the directory tree
where ESGET data is stored. Used to find files, examine paths, and move and
rename files."""

import logging
import datetime
import os
import re
import hashlib
import shutil
import glob
from time import ctime


class EsgetFS(object):
    '''Handles filesystem operations inside the
    ESGET storage directory-tree'''

    def __init__(self, config):
        self.fileroot = config.get('DEFAULT', 'fileroot')
        self._log = logging.getLogger(self.__class__.__name__)
        self.pathfields = config.get('DEFAULT', 'pathfields').split(',')
        self.dbname = config.get('Paths', 'dbname')
        self.wget_tmp_dir = config.get('Paths', 'wget_dir')
        self.wget_log_dir = config.get('Paths', 'wget_log_dir')
        self.tmpdir = config.get('Paths', 'tmpdir')
        self.logfile = config.get('Paths', 'logfile')
        self.publicdbpath = os.path.join(self.fileroot,
                                         config.get('Tuning', 'publicdbdir'),
                                         os.path.basename(self.dbname))
        self.no_storagefiles = config.get('Tuning',
                                          'no_storagefiles').split(',')

    def unlink(self, ulfiles):
        '''Moves files to be unliked into special subdirectory'''
        if len(ulfiles) == 0:
            self._log.info("No files to unlink")
            return()
        curdate = datetime.date.today().isoformat()
        unlinkpath = os.path.join(self.fileroot, "unlinked", "{0}"
                                  .format(curdate))
        os.umask(0022)
        if not os.path.exists(unlinkpath):
            self._log.info("Making unlink - directory {0}".format(unlinkpath))
            os.makedirs(unlinkpath, 0755)
        self._log.info("Moving {0} files into {1}"
                       .format(len(ulfiles), unlinkpath))
        for f in ulfiles:
            fulpath = os.path.join(unlinkpath,
                                   os.path.relpath(f, self.fileroot))
            try:
                os.renames(f, fulpath)
            except:
                self._log.error("Could not unlink {0}.".format(f))
        return()

    def relink(self, unlinkdir):
        '''Moves previously unliked files back into main storage-tree'''
        unlinkpath = os.path.join(self.fileroot, "unlinked", unlinkdir)
        if not os.path.exists(unlinkpath):
            self._log.error("Unlink-directory {0} does not exists." +
```

Variable explorer

Name	Type	Size	Value
e	float	1	2.718281828459045
euler_gamma	float	1	0.5772156649015329
pi	float	1	3.141592653589793

Object inspector | Variable explorer | File explorer | Breakpoints

Console

Python 1 00:03:01

```
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.

Imported NumPy 1.8.2, SciPy 0.14.0, Matplotlib 1.4.0
Type "scientific" for more details.
>>> runfile(r'/home/hvwaldow/esget/code/esget_fs.py', wdir=r'/home/hvwaldow/esget/code')
>>> """ esget_fs implements functions that operate mainly in the directory tree
... where ESGET data is stored. Used to find files, examine paths, and move and
... rename files."""
' esget_fs implements functions that operate mainly in the directory tree\nwhere ESGET data is stored. Used to find files, ex
amine paths, and move and\nrename files.'
>>>
>>> import logging
>>> import datetime
>>> import os
>>> import re
>>> import hashlib
>>> import shutil
>>> import glob
>>> from time import ctime
>>>
```

Console | History log

Permissions: RW End-of-lines: LF Encoding: UTF-8-GUESSED Line: 7 Column: 10 Memory: 34 %

Python development tools

Editors with plugins

emacs@mie.ethz.ch

File Edit Options Buffers Tools Python Help

```
114     print("Scanning of {0} files finished in {1:.2f} hours"
115           .format(len(paths), t1))
116
117
118 def mk_fullrecordlist(local_fs_record, pathfields):
119     '''Expands each filename of recordlist into fields
120     associated with the filepath. If class attribute "recordlist"
121     (which is usually provided by scan.filesystem()) is empty,
122     an attempt is made to read the file <local_fs_record>'''
123
124     def split_path(path):
125         ''' Recursive function to create a list of path elements'''
126         folders = list()
127         if path == '' or path == '/':
128             return([path])
129         else:
130             rest, folder = os.path.split(path)
131             folders = split_path(rest)
132             folders.append(folder)
133             return(folders)
134
135     try:
136         fh = open(local_fs_record, "r")
137         recordlist = cPickle.load(fh)
138         fh.close()
139     except:
140         print("Could not read {0}. Aborting"
141               .format(local_fs_record))
142         sys.exit(1)
143!sys.
144     path statement: path = ['', '/home/hvwaldow/.emacs.d/.python-environ s
145     stderr                                  statement: stderr = open() s
146     stdout                                  statement: stdout = open() s
147     argv statement: argv = ['/home/hvwaldow/.emacs.d/.python-environment s
148     mro                                                function: sys.mro f
149     lib                                       statement: lib = str() s
150     arch                                     statement: arch = str() s
151     exit                                          function: sys.exit f
152     stdin                                   statement: stdin = open() s
153     flags                                statement: flags = sys.flags() s
154!# paths = [os.path.join(PREFIX, x[1].lstrip('/'), os.path.basename(x[0])) for
   ! x in fromto]
155 # allthere = all([os.path.exists(x) for x in paths])
156 # print("All there: {0}".format(allthere))
157 # scan_filesystem(paths, local_fs_record)
158 recordlist = mk_fullrecordlist(local_fs_record, pathfields)
159 C.update_files(recordlist)
-:**-example.py     32% (143,4)     (Python AC FlyC:11/1)
E901 SyntaxError: invalid syntax
```

```
In [13]: pathfields = config.get('DEFAULT', 'pathfields').split(',')


In [14]:
In [14]: pathfields
Out[14]:
['domain',
 'experiment',
 'time_frequency',
 'variable',
 'institute',
 'driving_model',
 'model',
 'ensemble',
 'version']

In [15]:
U:**-*Python*      Bot (53,9)     (Inferior Python:run Shell-Compile)
```

Python development tools

Editors with plugins

- Emacs
- Vim
- other editors with varying degrees of support
- Geany, TextWrangler, . . .
- More screen-estate for code & interpreter
- Keyboad instead of mouse ⇒ faster!
- Might be complicated to set up, initially.
- Steeper learning curve
- If you already use Emacs or Vim, go for it!

Python development tools

Editors with plugins

- Emacs
- Vim
- other editors with varying degrees of support
- Geany, TextWrangler, . . .
- More screen-estate for code & interpreter
- Keyboad instead of mouse ⇒ faster!
- Might be complicated to set up, initially.
- Steeper learning curve
- If you already use Emacs or Vim, go for it!

Python development tools

Editors with plugins

- Emacs
- Vim
- other editors with varying degrees of support
- Geany, TextWrangler, ...
- More screen-estate for code & interpreter
- Keyboad instead of mouse ⇒ faster!
- Might be complicated to set up, initially.
- Steeper learning curve
- If you already use Emacs or Vim, go for it!

Python development tools

And now for something completely different . . .

IP[y]: Notebook Notebook_1

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

Learning and teaching Python in the 21st century

In [6]:
```
from IPython.display import IFrame
IFrame('http://www.gfdl.noaa.gov/blog/isaac-held/', width='100%', height=300)
```

Out[6]:

50. Volcanoes and the Transient Climate Response - Part II

Posted on September 3, 2014 by Isaac Held