

# C2SM workshop

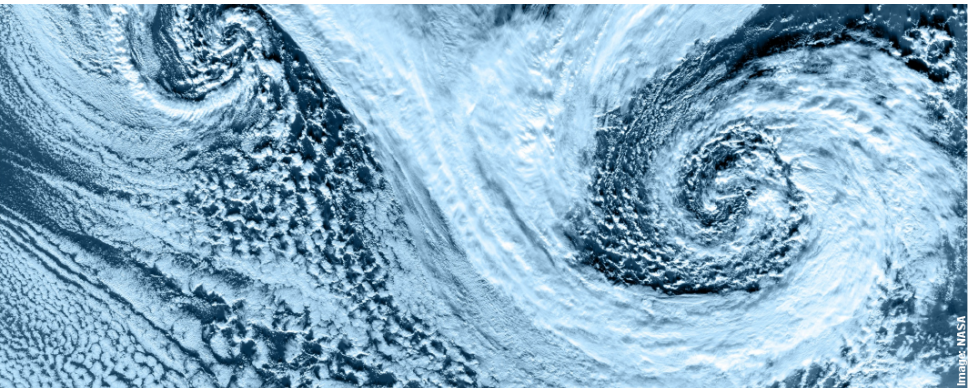
## Scientific Programming in Python

Harald von Waldow (C2SM/ETHZ)

Nicolas Piaget (Atm.Dyn./ETHZ)

Timm Gross (GIUB/UNIBE)

2015-02-11

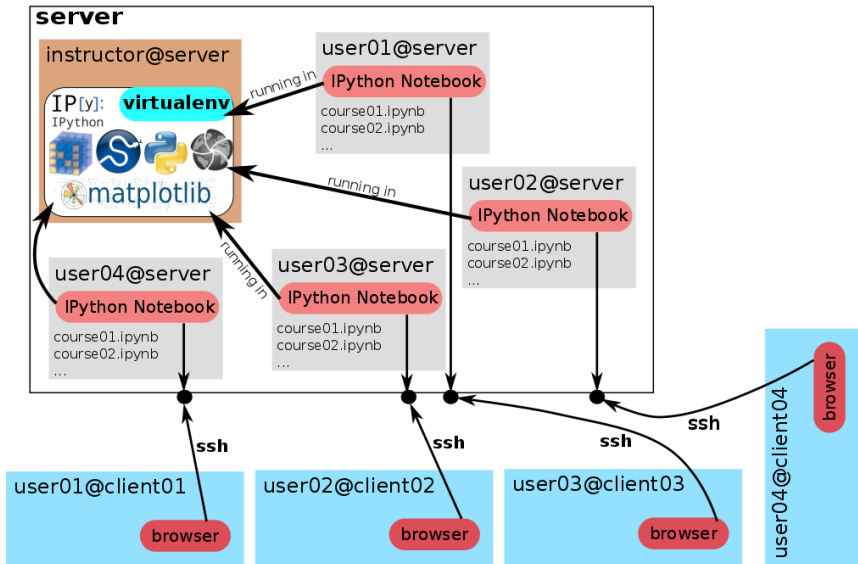


# Schedule

09:30	Welcome, organizational matters	Harald
09:45	Python — The big picture	Harald
10:00	Python development tools	Harald
10:15	<b>Exercise 1:</b> The IPython Notebook	Harald/Nicolas
10:45	Coffee Break <b>15 min</b>	
11:00	<b>Exercise 2:</b> Basic Python	Harald
12:00	Lunch Break <b>1 h</b>	
13:00	<b>Exercise 3:</b> Numpy, Scipy, Matplotlib, netCDF4	Nicolas
14:30	Coffee Break <b>15 min</b>	
14:45	<b>Exercise 4:</b> More fun with trajectory plots	Nicolas
15:45	Coffee Break <b>15 min</b>	
16:00	<b>Exercise 5:</b> 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:
  - functional, object-oriented
  - procedural 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-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-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-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

## 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-oriented
  - 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-oriented
  - 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-oriented
  - 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-oriented
  - 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)
- You 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)
- You 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)
- You 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)
- You 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)
- You 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)
- You 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)
- You 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)
- You 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.
- ...

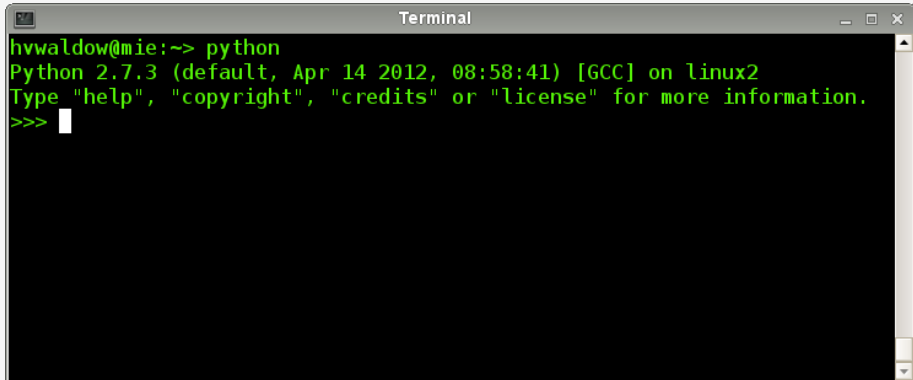
Python programmers strive to write *pythonic*

- 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

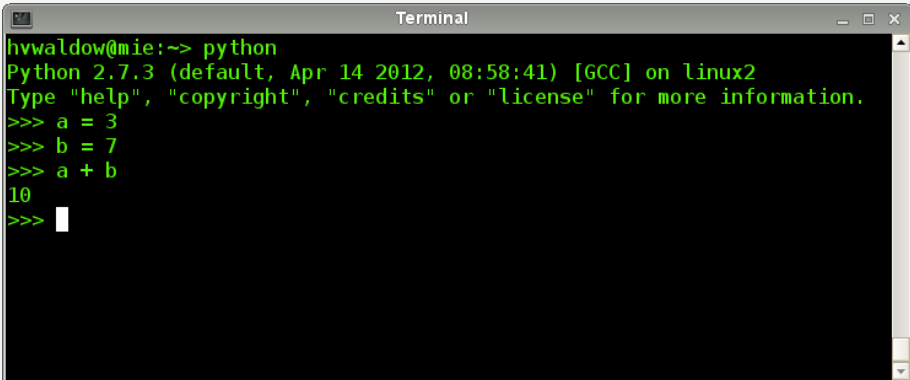
A terminal window titled "Terminal" with standard window controls. The text inside is green on a black background. It shows the command 'python' being executed, followed by the Python 2.7.3 startup banner and the prompt '>>>' with a cursor.

```
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

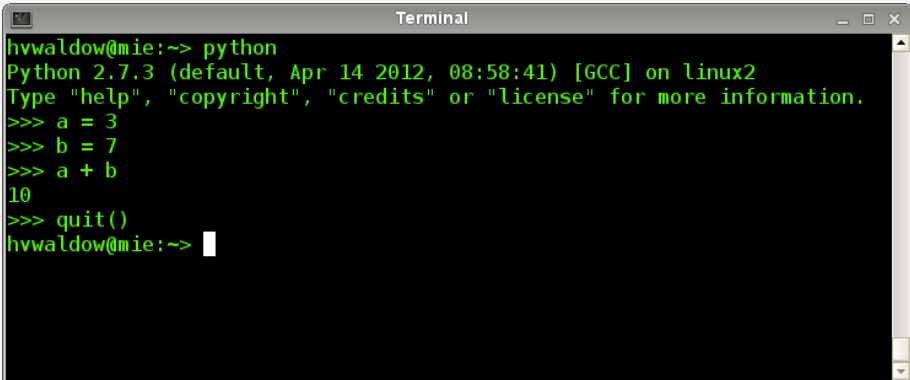
A screenshot of a terminal window titled "Terminal". The prompt is "hvwaldow@mie:~>". The user has entered "python", which has started the Python 2.7.3 interpreter. The interpreter displays its version and build information: "Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2". It then prompts the user to type "help", "copyright", "credits", or "license" for more information. The user has entered three lines of code: ">>> a = 3", ">>> b = 7", and ">>> a + b". The interpreter has output "10" for the last line. A cursor is visible on the line ">>>".

```
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

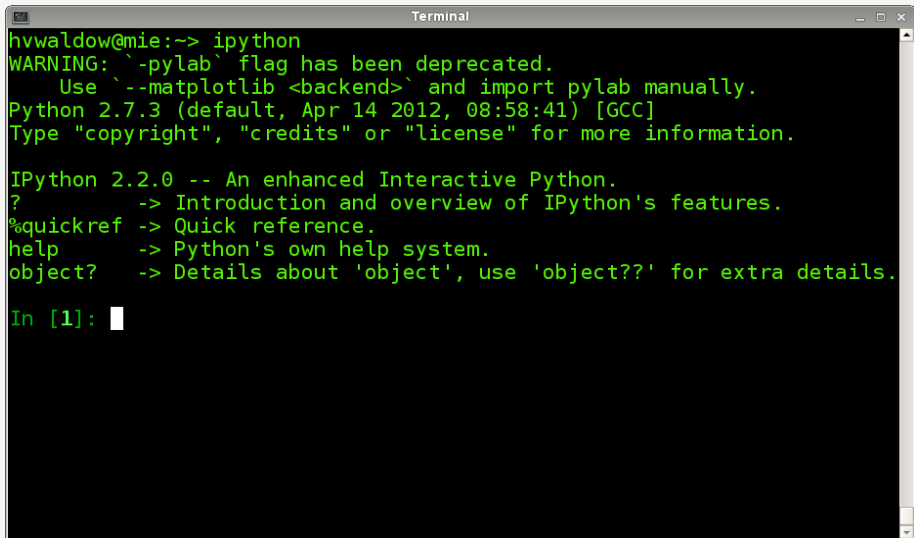
A terminal window titled "Terminal" with standard window controls (minimize, maximize, close) in the top right corner. The terminal shows a user at a prompt "hvwaldow@mie:~>" typing "python". The output shows "Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2" followed by a message to type "help", "copyright", "credits", or "license" for more information. The user then enters three lines of Python code: ">>> a = 3", ">>> b = 7", and ">>> a + b". The output of the last line is "10". Finally, the user enters ">>> quit()", and the prompt returns to "hvwaldow@mie:~>".

```
Terminal
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

A terminal window titled "Terminal" with standard window controls (minimize, maximize, close) in the top right corner. The terminal displays the output of running the 'ipython' command. The text is as follows:

```
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.

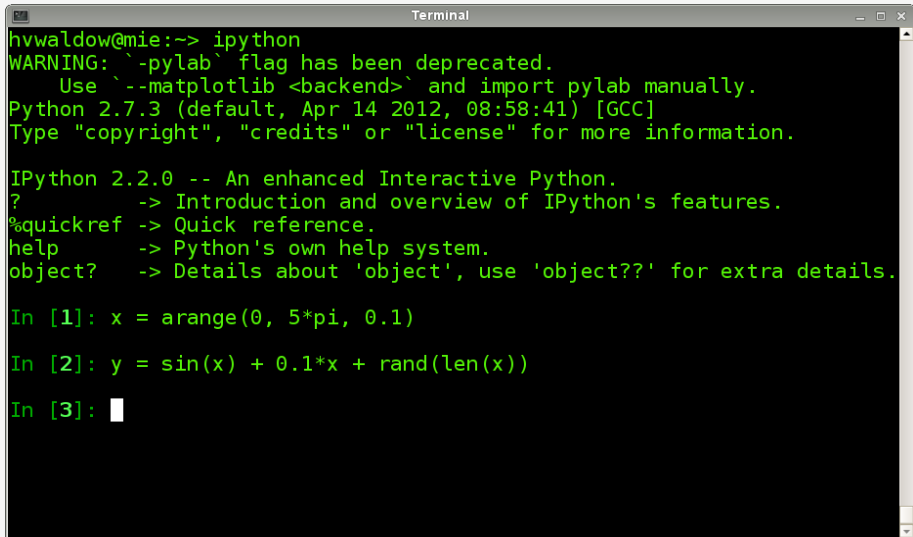
IPython 2.2.0 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref   -> Quick reference.
help        -> Python's own help system.
object?     -> Details about 'object', use 'object??' for extra details.

In [1]: █
```

Typing “ipython” will call the IPython shell.

# Basic Python

## IPython shell



```
Terminal
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)

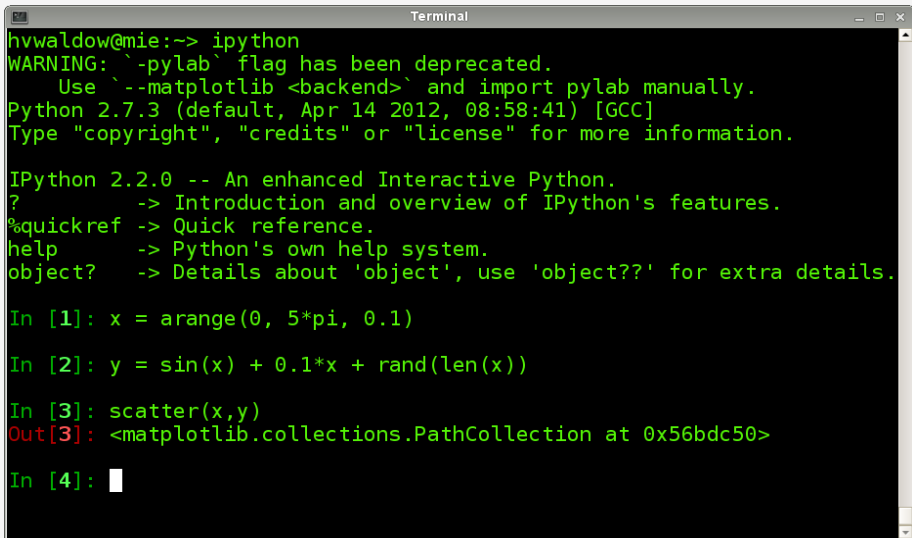
In [2]: y = sin(x) + 0.1*x + rand(len(x))

In [3]:
```

Math functions “magically” loaded.

# Basic Python

## IPython shell

A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal shows the execution of the 'ipython' command. It displays a warning about the deprecated '-pylab' flag, the Python version (2.7.3), and the IPython version (2.2.0). It lists several help options: '?', '%quickref', 'help', and 'object?'. The user then enters four lines of code: 'x = arange(0, 5\*pi, 0.1)', 'y = sin(x) + 0.1\*x + rand(len(x))', 'scatter(x,y)', and an empty prompt 'In [4]:'. The output for the third line is '<matplotlib.collections.PathCollection at 0x56bdc50>'.

```
Terminal
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))

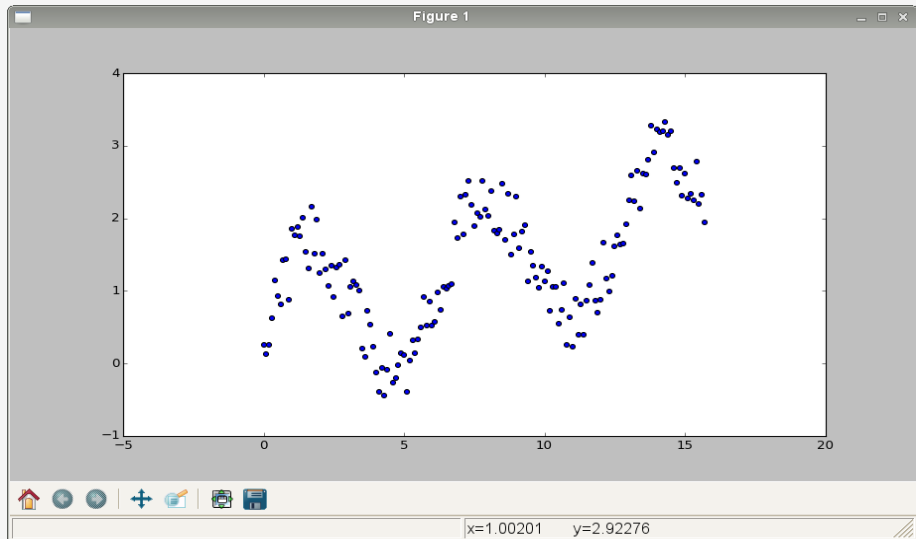
In [3]: scatter(x,y)
Out[3]: <matplotlib.collections.PathCollection at 0x56bdc50>

In [4]:
```

Plot functions as well.

# Basic Python

## IPython shell



# Basic Python

## IPython shell

```
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)

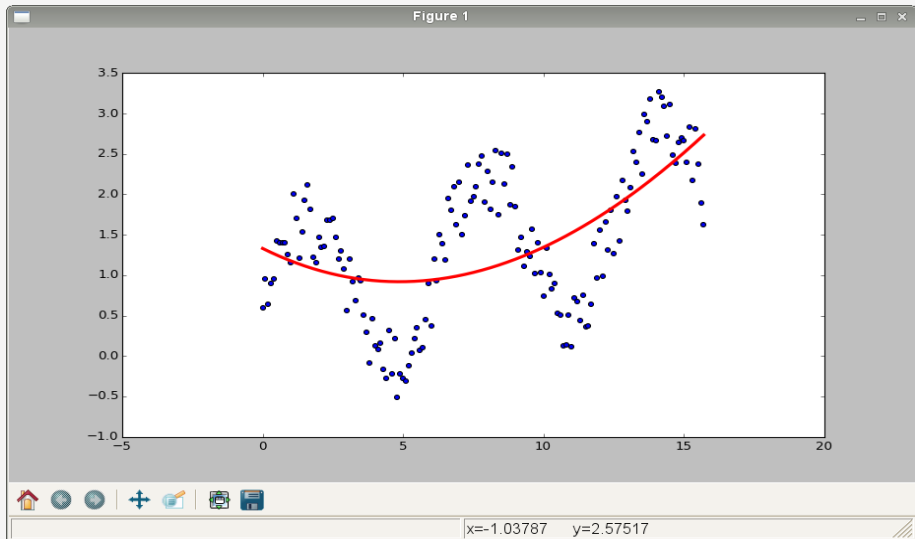
In [5]: plot(x, polyval(fit,x), linewidth=3, c="red")
Out[5]: [<matplotlib.lines.Line2D at 0x5959bd0>]

In [6]:
```



# Basic Python

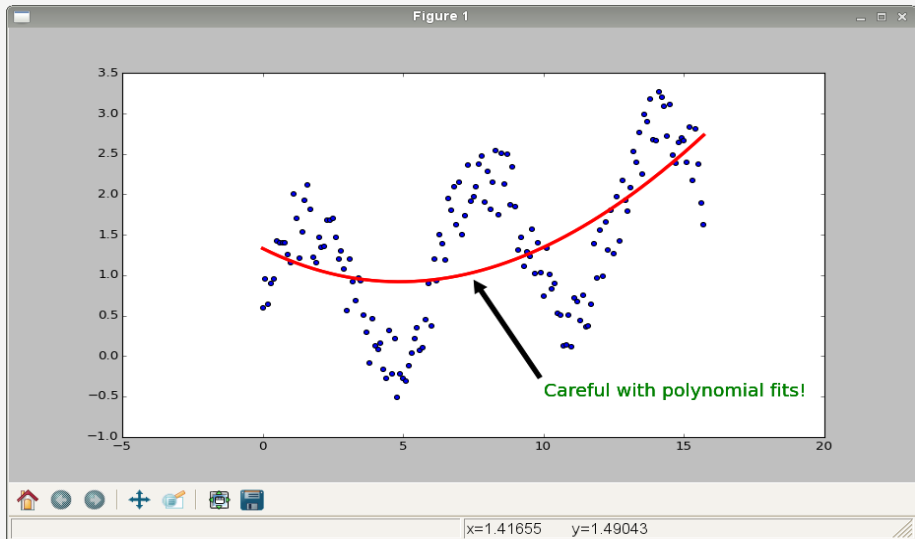
## IPython shell



New content can be added interactively.

# Basic Python

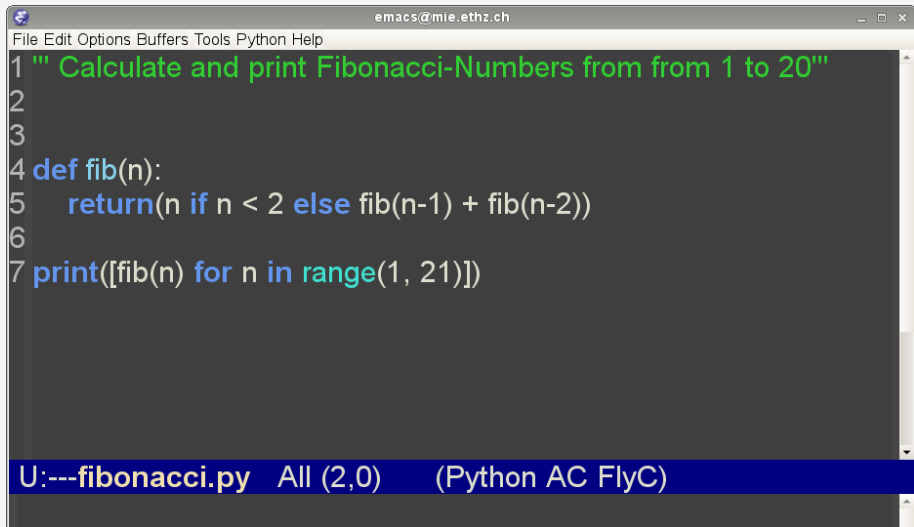
## IPython shell



New content can be added interactively.

# Basic Python

## Run a python script



The screenshot shows an Emacs editor window titled 'emacs@mie.ethz.ch'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', and 'Help'. The code is as follows:

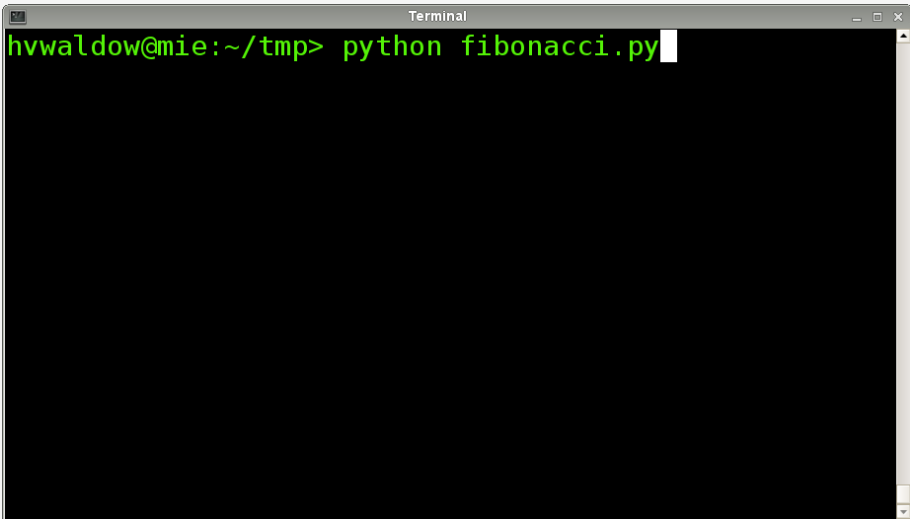
```
1 """ Calculate and print Fibonacci-Numbers from from 1 to 20"""
2
3
4 def fib(n):
5     return(n if n < 2 else fib(n-1) + fib(n-2))
6
7 print([fib(n) for n in range(1, 21)])
```

The status bar at the bottom displays 'U:---fibonacci.py All (2,0) (Python AC FlyC)'.

Save file with ending ".py"

# Basic Python

Run a python script

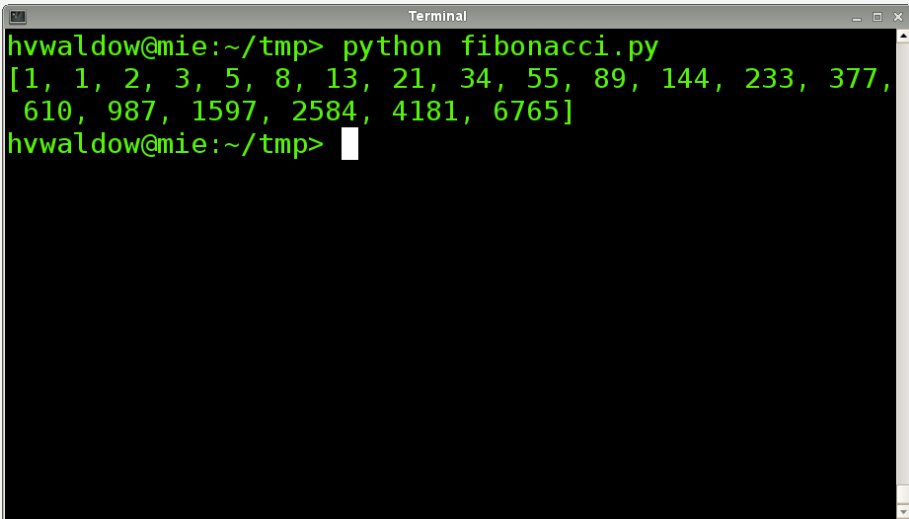
A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal has a black background and green text. The prompt is "hvwaldow@mie:~/tmp>". The command "python fibonacci.py" has been entered, and a white cursor is positioned at the end of the command.

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

Call python interpreter with filename as argument.

# Basic Python

Run a python script

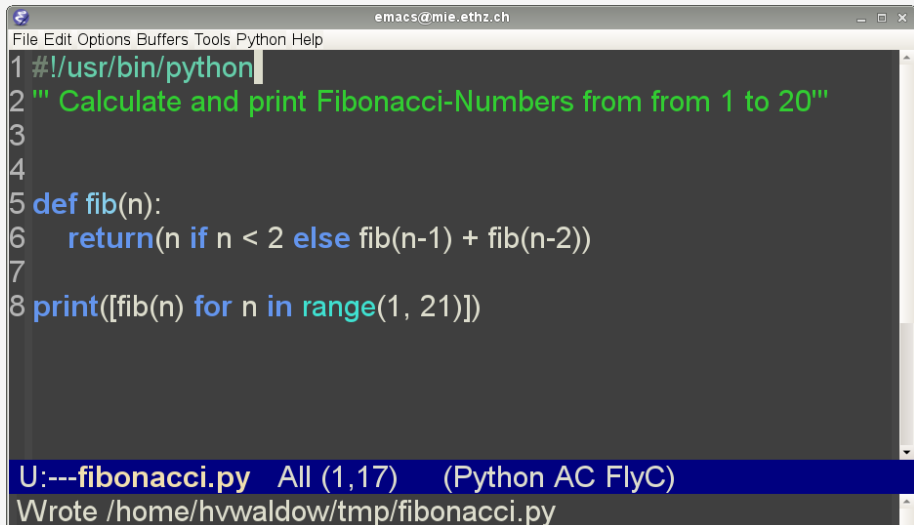
A terminal window titled "Terminal" with standard window controls (minimize, maximize, close) in the top right corner. The prompt is "hvwaldow@mie:~/tmp>". The command "python fibonacci.py" has been entered and executed. The output is a single line of green text: "[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765]". The prompt "hvwaldow@mie:~/tmp>" is followed by a white cursor block.

```
Terminal
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



The screenshot shows an Emacs editor window titled 'emacs@mie.ethz.ch'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', and 'Help'. The code is as follows:

```
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)])
```

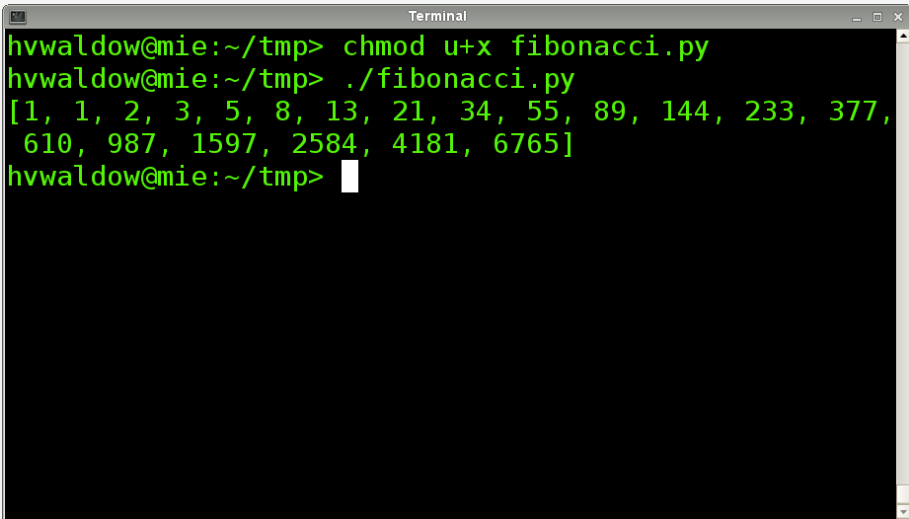
At the bottom of the window, a status bar displays the following information:

U:---fibonacci.py All (1,17) (Python AC FlyC)  
Wrote /home/hvwaldow/tmp/fibonacci.py

“Shebang” syntax works.

# Basic Python

## Run a python script

A terminal window titled "Terminal" with standard window controls. The prompt is "hvwaldow@mie:~/tmp". The user enters "chmod u+x fibonacci.py" and then "./fibonacci.py". The output is a list of Fibonacci numbers: "[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765]". The prompt returns to "hvwaldow@mie:~/tmp>".

```
Terminal
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

The screenshot displays the Spyder Python IDE interface. The main window is titled "Spyder (Python 2.7)". The top menu bar includes File, Edit, Search, Source, Run, Interpreters, Tools, and View. Below the menu is a toolbar with various icons for file operations and running code.

The left pane shows the Editor with a file named `temp.py` containing Python code. The code defines a class `EsetFS` for handling filesystem operations. The code is as follows:

```
1''' eset fs implements functions that operate mainly in the directory tree
2where ESET data is stored. Used to find files, examine paths, and move and
3rename files.'''
4
5import logging
6import datetime
7import os
8
9import logging
10import datetime
11import os
12import glob
13from time import time
14
15class EsetFS(object):
16    '''Handles filesystem operations inside the
17    ESET storage directory-tree'''
18
19    def __init__(self, config):
20        self.fileroot = config.get('DEFAULT', 'fileroot')
21        self.log = logging.getLogger(self.__class__.__name__)
22        self.pathfields = config.get('DEFAULT', 'pathfields').split(',')
23        self.dbname = config.get('Paths', 'dbname')
24        self.wget_tap_dir = config.get('Paths', 'wget_dir')
25        self.wget_log_dir = config.get('Paths', 'wget_log_dir')
26        self.tapdir = config.get('Paths', 'tapdir')
27        self.logfile = config.get('Paths', 'logfile')
28        self.publicdbpath = os.path.join(self.fileroot,
29                                         config.get('Tuning', 'publicdbdir'),
30                                         os.path.basename(self.dbname))
31        self.no_storagefiles = config.get('Tuning',
32                                         'no_storagefiles').split(',')
33
34    def unlink(self, ufiles):
35        '''Moves files to be unlinked into special subdirectory'''
36        if len(ufiles) == 0:
37            self.log.info("No files to unlink")
38            return()
39        curdate = datetime.date.today().isoformat()
40        unlinkpath = os.path.join(self.fileroot, "unlinked", "%02"
41                                  .format(curdate))
42        os.makedirs(unlinkpath, 0755)
43        if not os.path.exists(unlinkpath):
44            self.log.info("Moving (%d) files into (%s)"
45                          .format(len(ufiles), unlinkpath))
46        self._log.info("Moving (%d) files into (%s)"
47                      .format(len(ufiles), unlinkpath))
48        for f in ufiles:
49            fulpath = os.path.join(unlinkpath,
50                                  os.path.relpath(f, self.fileroot))
51            try:
52                os.rename(f, fulpath)
53            except:
54                self._log.error("Could not unlink (%s)." % f)
55        return()
56
57    def relink(self, unlinkedir):
58        '''Moves previously unlinked files back into main storage-tree'''
59        unlinkpath = os.path.join(self.fileroot, "unlinked", unlinkedir)
60        if not os.path.exists(unlinkpath):
61            self._log.error("Unlinked directory (%s) does not exist" % unlinkpath)
```

The right pane shows the Variable explorer with a table of variables:

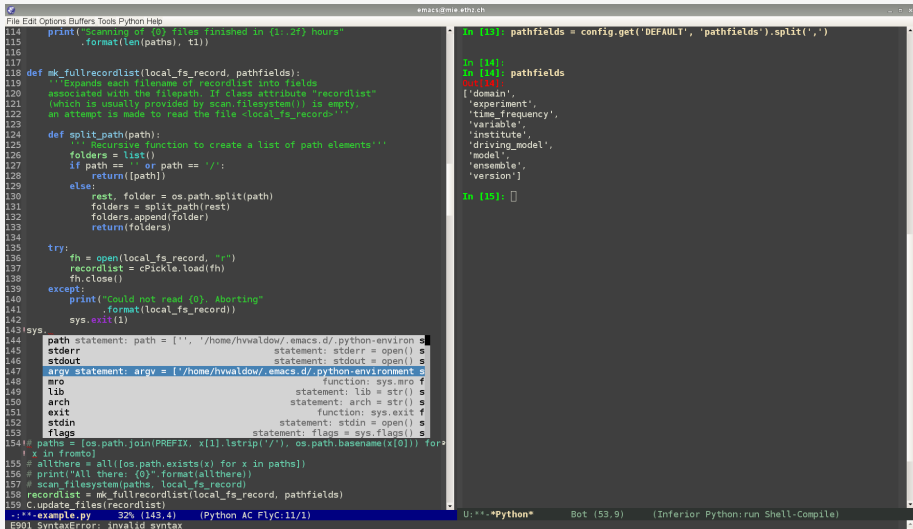
Name	Type	Size	Value
e	float	1	2.718281828459045
euler_gamma	float	1	0.5772156649015329
pi	float	1	3.141592653589793

Below the Variable explorer is the Object inspector, which shows the File explorer, Breakpoints, and Console. The Console shows the output of the Python interpreter, including the version (Python 2.7.3) and the path to the ESET data directory.

At the bottom of the IDE, the status bar shows the following information: Permissions: **RN**, End-of-lines: **LF**, Encoding: **UTF-8-GUESSED**, Line: **7**, Column: **10**, Memory: **34 %**.

# Python development tools

## Editors with plugins



The screenshot shows the Emacs editor interface with a Python script on the left and its execution output on the right. The script defines a function to scan a directory for files and a function to split a path into folders. The output shows the execution of the script, including the path of the current file and the list of files found.

```
File Edit Options Buffers Tools Python Help
114 print("Scanning of {} files finished in {:.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 {}. Aborting"
141               .format(local_fs_record))
142         sys.exit(1)
143
144 sys.path.append(os.path.dirname(os.path.abspath(__file__)))
145
146 path statement: path = [' ', '/home/hvwaldow/.emacs.d/python-environ s
147 stderr statement: stderr = open() s
148 stdout statement: stdout = open() s
149 argv statement: argv = ['/home/hvwaldow/.emacs.d/python-environment s
150 mro function: sys.mro f
151 lib statement: lib = str() s
152 arch statement: arch = str() s
153 exit function: sys.exit f
154 statement: stdin = open() s
155 statement: stdout = open() s
156 statement: stderr = open() s
157 statement: flags = sys.flags() s
158
159 paths = [os.path.join(PREFIX, x[1].lstrip('/'), os.path.basename(x[0])) for
160           'x in fromto]
161
162 # all there = all([os.path.exists(x) for x in paths])
163 # print("All there: {}").format(allthere)
164 # scan.filesystem(paths, local_fs_record)
165 recordlist = mk_fullrecordlist(local_fs_record, pathfields)
166 C.update_files(recordlist)
167
168 ***-example.py 32% (143,4) (Python AC FlyC:11/1)
169
170 In [13]: pathfields = config.get('DEFAULT', 'pathfields').split(',')
171
172 In [14]:
173 Out[14]:
174 ['domain',
175  'experiment',
176  'time_frequency',
177  'variable',
178  'institute',
179  'driving_model',
180  'model',
181  'ensemble',
182  'version']
183
184 In [15]:
```

# Python development tools

## Editors with plugins

- Emacs
- Vim
- other editors with varying degrees of support
- Geany, TextWrangler, ...
- More screen-estate for code & interpreter
- Keyboard instead of mouse  $\Rightarrow$  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
- Keyboard instead of mouse  $\Rightarrow$  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
- Keyboard instead of mouse  $\Rightarrow$  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 ...



# 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](#)

