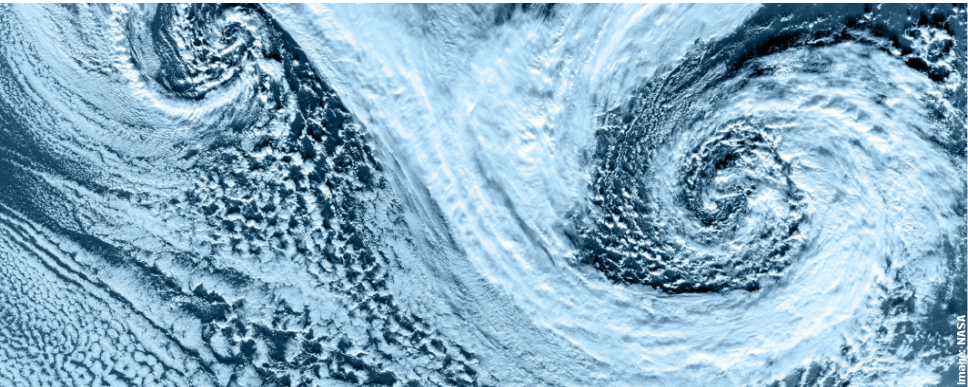# C2SM workshop
Scientific Programming in Python

Harald von Waldow (C2SM)   Bas Crezee (IAC/Atm.Dyn.)

Nicolas Piaget (IAC/Atm.Dyn.) Marina Dütsch (IAC/Atm.Dyn.)

2014–09–11

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

Room

- Is not going top be locked during breaks.
- Take your valuables with you.
- Logout or lock screen

Organizational Matters

Room

- Is not going top be locked during breaks.
- Take your valuables with you.
- Logout or lock screen

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 `iacdipl-3.ethz.ch`. You should not need to worry about that.

Organizational Matters

Login

- username: `c2smXX`
- XX is 02 ... 19
- password: `-- censored --`
- After login wait 10 sec or so ...
- In case of password-asking pop-up window: Click"cancel".
- 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:

Python — the big picture

Classification

- interpreted
- general-purpose, high-level
- dynamically typed
- garbage-collection
- multi-paradigm:

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-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
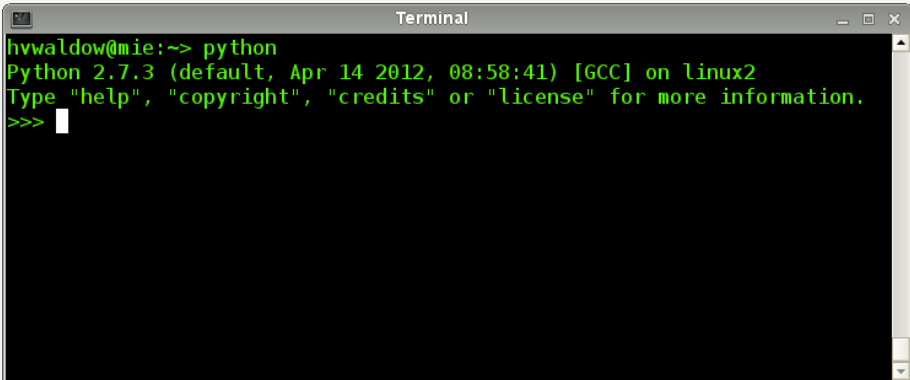
Philosophy

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
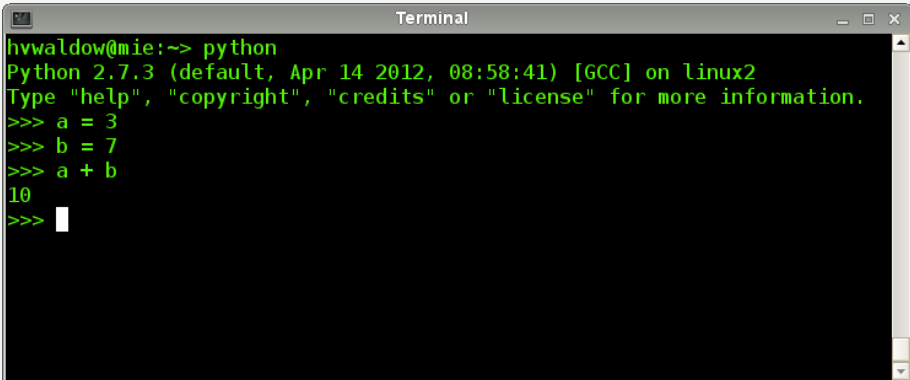
Typing "python" will call the "raw" Python shell.

Basic Python

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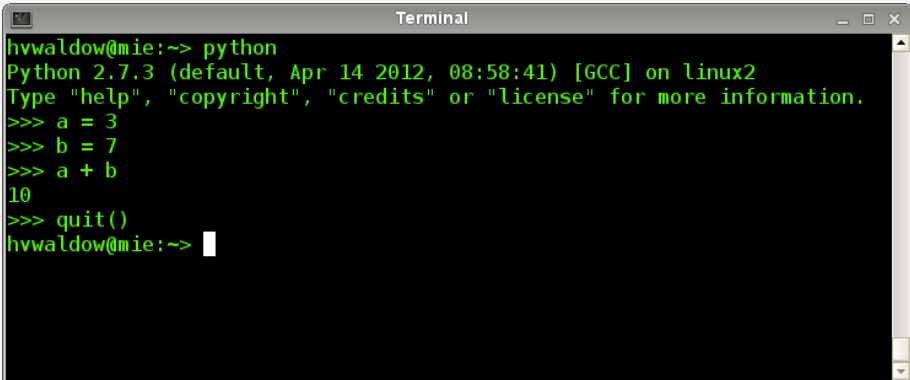
```
hvwaldow@mie:~> python
Python 2.7.3 (default, Apr 14 2012, 08:58:41) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> b = 7
>>> a + b
10
>>>
```

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
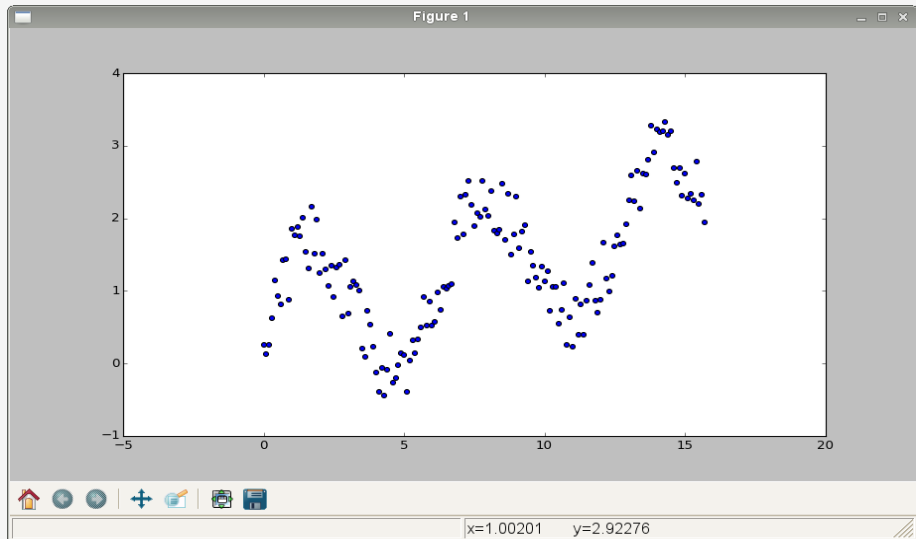
Basic Python

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
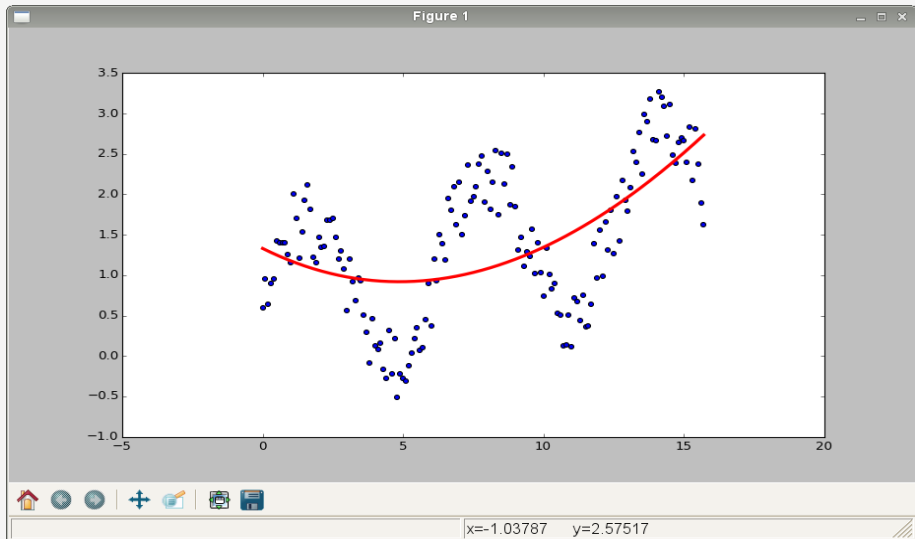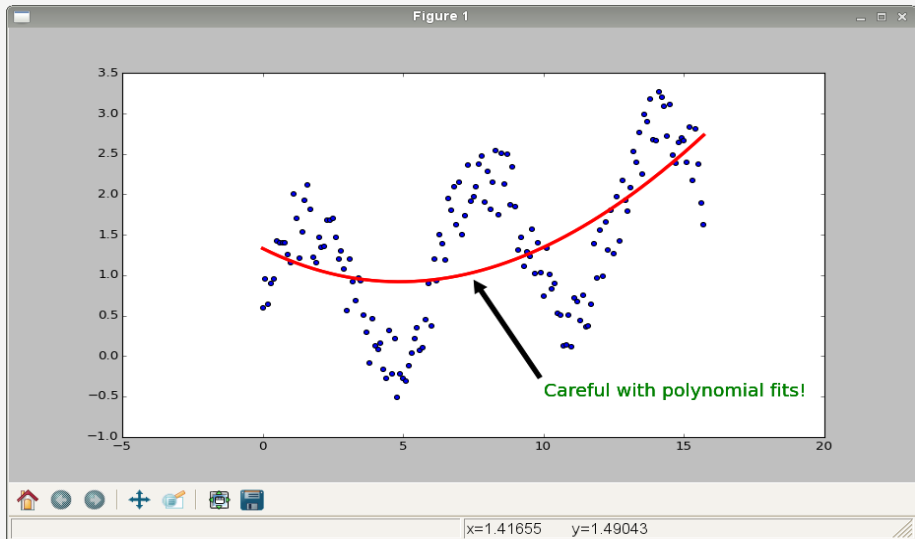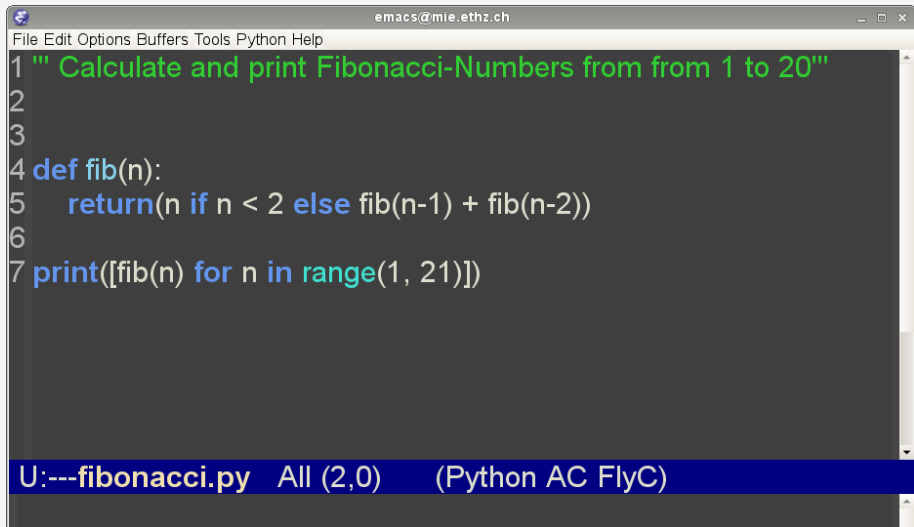
New content can be added interactively.

Basic Python

New content can be added interactively.

Basic Python

Run a python script

File Edit Options Buffers Tools Python Help
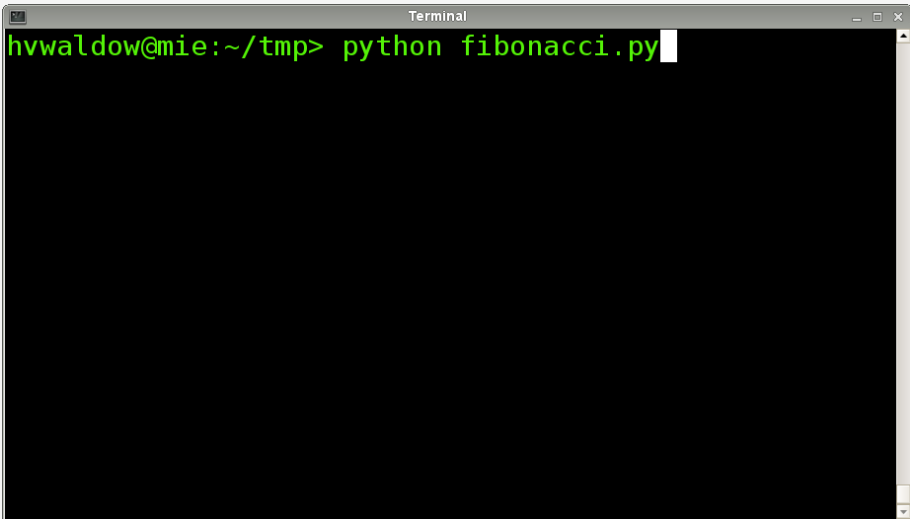
```python
''' Calculate and print Fibonacci-Numbers from from 1 to 20'''


def fib(n):
    return(n if n < 2 else fib(n-1) + fib(n-2))

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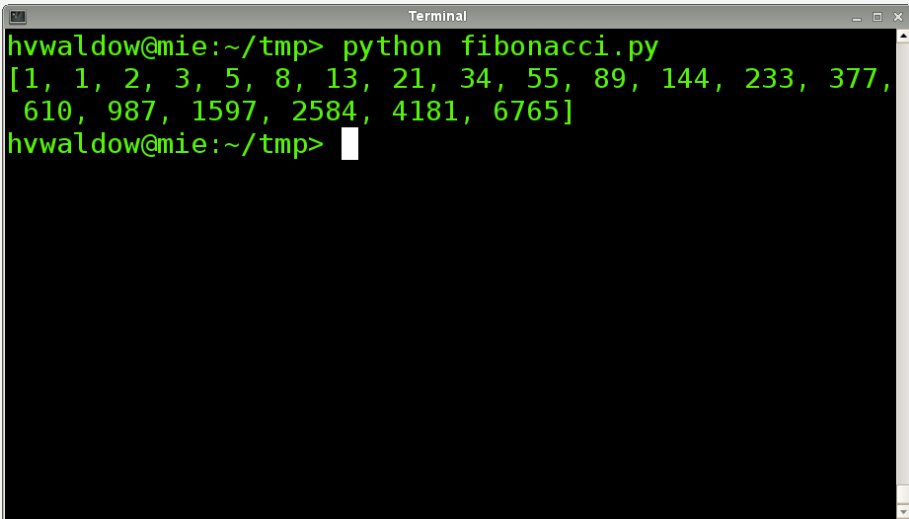
```
hvwaldow@mie:~/tmp> python fibonacci.py
```

Call python intepreter with filename as argument.

Basic Python

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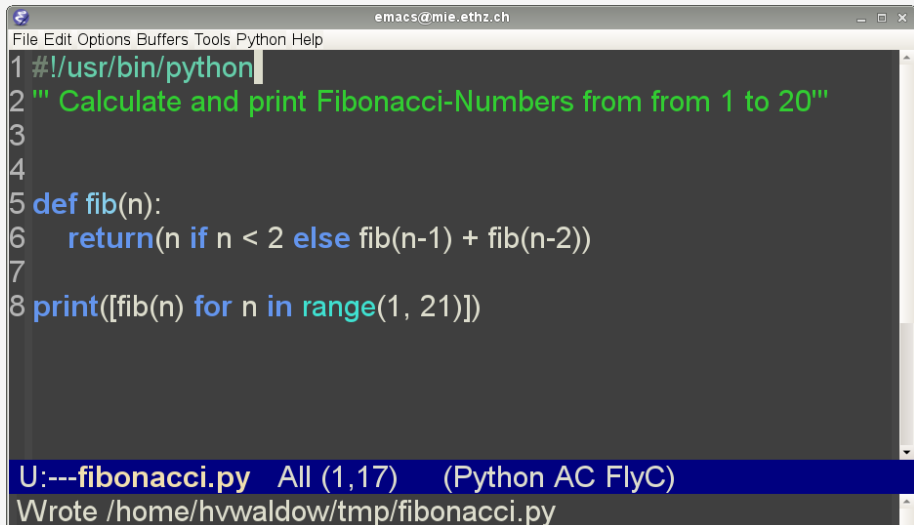
```
hvwaldow@mie:~/tmp> python fibonacci.py
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
 610, 987, 1597, 2584, 4181, 6765]
hvwaldow@mie:~/tmp>
```

Output goes to stdout.

Basic Python

Run a python script

```python
#!/usr/bin/python
''' Calculate and print Fibonacci-Numbers from from 1 to 20'''


def fib(n):
    return(n if n < 2 else fib(n-1) + fib(n-2))

print([fib(n) for n in range(1, 21)])
```
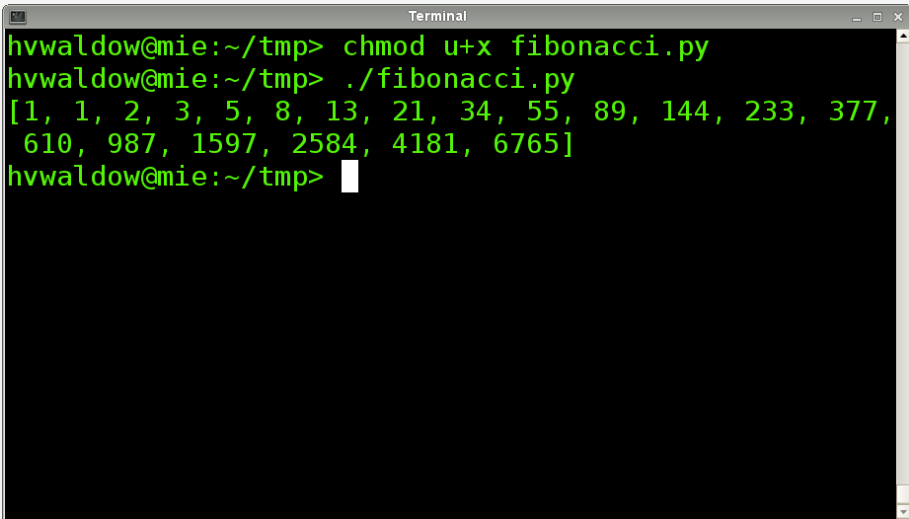
U:---**fibonacci.py**   All (1,17)   (Python AC FlyC)

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

Python development tools

Editors with plugins

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 ⇒ 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 ⇒ 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

IP[y]: Notebook Notebook_1

Code ▾   Cell Toolbar: None ▾

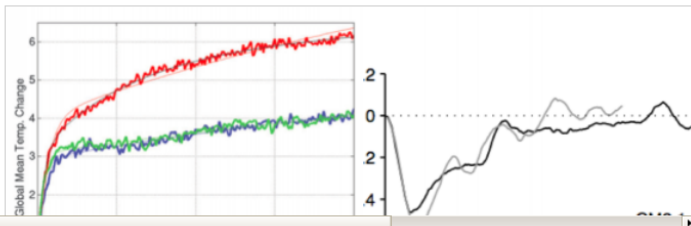## Learning and teaching Python in the 21st century

In [6]:
```python
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

Mr. Crezee, übernehmen Sie!