A brief tour of the IPython notebook

Rendered by nbconvert using **Reveal.js**!

by Damián Avila

Loading [MathJax]/jax/output/HTML-CSS/jax.js

This document will give you a brief tour of the capabilities of the IPython notebook.
You can view its contents by scrolling around, or execute each cell by typing `Shift-Enter`. After you conclude this brief high-level tour, you should read the accompanying notebook titled `01_notebook_introduction`, which takes a more step-by-step approach to the features of the system.
The rest of the notebooks in this directory illustrate various other aspects and capabilities of the IPython notebook; some of them may require additional libraries to be executed.

NOTE: This notebook *must* be run from its own directory, so you must `cd` to this directory and then start the notebook, but do *not* use the `--notebook-dir` option to run it from another location.

The first thing you need to know is that you are still controlling the same old IPython you're used to, so things like shell aliases and magic commands still work:

In [1]:
```
pwd
```

Out[1]:
```
u'/home/damian/Desarrollos/ipython_mtaui_slide'
```

In [2]:
```
ls
```

```
COPYING.txt              IPython/     python-logo.svg
setupbase.py  setup.py*
docs/                    ipython.py*  README.rst
setupegg.py*  tools/
example_nb_tour.ipynb  MANIFEST.in  scripts/
setupext/     tox.ini
```

In [3]:

```
message = 'The IPython notebook is great!'
# note: the echo command does not run on
Windows, it's a unix command.
!echo $message
```

```
The IPython notebook is great!
```

Plots with matplotlib

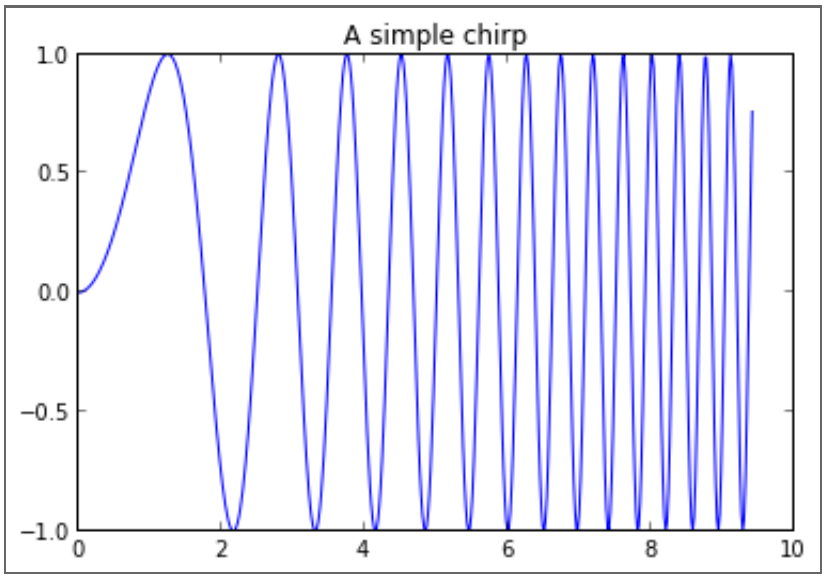
IPython adds an 'inline' matplotlib backend, which embeds any matplotlib figures into the notebook.

In [4]:
```
%pylab inline
```

```
Welcome to pylab, a matplotlib-based Python environment
[backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [5]:

```
x = linspace(0, 3*pi, 500)
plot(x, sin(x**2))
title('A simple chirp');
```



You can paste blocks of input with prompt markers, such as those from **the official Python tutorial**

In [6]:
```
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
```

```
Be careful not to fall off!
```

Errors are shown in informative ways:

In [7]:
```
%run non_existent_file
```

```
ERROR: File `u'non_existent_file.py'` not found.
```

In [8]:

```
x = 1
y = 4
z = y/(1-x)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-8-dc39888fd1d2> in <module>()
      1 x = 1
      2 y = 4
----> 3 z = y/(1-x)

ZeroDivisionError: integer division or modulo by zero
```

When IPython needs to display additional information (such as providing details on an object via `x?` it will automatically invoke a pager at the bottom of the screen:

In [9]:
```
magic
```

Non-blocking output of kernel

If you execute the next cell, you will see the output arriving as it is generated, not all at the end.

In [10]:
```
import time, sys
for i in range(8):
    print i,
    time.sleep(0.5)
```

```
0 1 2 3 4 5 6 7
```

Clean crash and restart

We call the low-level system libc.time routine with the wrong argument via ctypes to segfault the Python interpreter:

In [*]:
```
import sys
from ctypes import CDLL
# This will crash a Linux or Mac system;
equivalent calls can be made on Windows
dll = 'dylib' if sys.platform == 'darwin'
else '.so.6'
libc = CDLL("libc.%s" % dll)
libc.time(-1)  # BOOM!!
```

Markdown cells can contain formatted text and code

You can *italicize*, **boldface**

- build
- lists

and embed code meant for illustration instead of execution in Python:

```
def f(x):
    """a docstring"""
    return x**2
```

or other languages:

```
if (i=0; i<n; i++) {
  printf("hello %d\n", i);
  x += 4;
}
```

Courtesy of MathJax, you can include mathematical expressions both inline: $e^{i\pi} + 1 = 0$ and displayed:

$$e^x=\sum_{i=0}^\infty \frac{1}{i!}x^i$$

Rich displays: include anyting a browser can show

Note that we have an actual protocol for this, see the `display_protocol` notebook for further details.

Images

An image can also be displayed from raw data or a url

In [12]:
```
Image(url='http://python.org/images/python-logo.gif')
```

Out[12]:

SVG images are also supported out of the box (since modern browsers do a good job of rendering them):

In [13]:
```
from IPython.display import SVG
SVG(filename='python-logo.svg')
```

Out[13]:

Embedded vs Non-embedded Images

As of IPython 0.13, images are embedded by default for compatibility with QtConsole, and the ability to still be displayed offline.

Let's look at the differences:

In [14]:

```
# by default Image data are embedded
Embed      = Image(
'http://scienceview.berkeley.edu/view/images/
newview.jpg')

# if kwarg `url` is given, the embedding is
assumed to be false
SoftLinked =
Image(url='http://scienceview.berkeley.edu/vi
ew/images/newview.jpg')

# In each case, embed can be specified
explicitly with the `embed` kwarg
# ForceEmbed =
Image(url='http://scienceview.berkeley.edu/vi
ew/images/newview.jpg', embed=True)
```

In [15]:
```
Embed
```

Out[15]:

In [16]:
```
SoftLinked
```

Out[16]:

Video

And more exotic objects can also be displayed, as long as their representation supports the IPython display protocol.

For example, videos hosted externally on YouTube are easy to load (and writing a similar wrapper for other hosted content is trivial):

In [17]:

```
from IPython.display import YouTubeVideo
# a talk about IPython at Sage Days at U. Washington, Seattle.
# Video credit: William Stein.
YouTubeVideo('1j_HxD4iLn8')
```

Out[17]:

Local Files

The above examples embed images and video from the notebook filesystem in the output areas of code cells. It is also possible to request these files directly in markdown cells if they reside in the notebook directory via relative urls prefixed with `files/`:

```
files/[subdirectory/]<filename>
```

For example, in the example notebook folder, we have the Python logo, addressed as:

```
<img src="files/python-logo.svg" />
```

External sites

You can even embed an entire page from another site in an iframe; for example this is today's Wikipedia page for mobile users:

In [19]:

```
HTML('<iframe
src=http://en.mobile.wikipedia.org/?
useformat=mobile width=700 height=350>
</iframe>')
```

Out[19]:

Search Wikipedia

Today's featured article

New York State Route 319 was a state highway in Chenango County, New York, in the United States. It was 5.47 miles (8.80 km) long and connected the hamlet of Preston to the nearby city of Norwich. What became NY 319 was originally built during the early 19th century as the privately owned Norwich and Preston Turnpike. The state of New York assumed ownership in the early 20th century, and the Preston–Norwich state highway was designated as NY 319 as part of the 1930 renumbering of state highways in New York. Maintenance of NY 319 was split between the state and the city of Norwich, with the New York State Department of Transportation handling the part of the route west of the city limits. In 1962

Mathematics

And we also support the display of mathematical expressions typeset in LaTeX, which is rendered in the browser thanks to the **MathJax library**.

Note that this is *different* from the above examples. Above we were typing mathematical expressions in Markdown cells (along with normal text) and letting the browser render them; now we are displaying the output of a Python computation as a LaTeX expression wrapped by the `Math()` object so the browser renders it. The `Math` object will add the needed LaTeX delimiters (`$$`) if they are not provided:

In [20]:
```
from IPython.display import Math
Math(r'F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k} dx')
```

Out[20]:

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k} dx$$

With the `Latex` class, you have to include the delimiters yourself. This allows you to use other LaTeX modes such as `eqnarray`:

In [21]:
```
from IPython.display import Latex
Latex(r"""\begin{eqnarray}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\,
\frac{\partial\vec{\mathbf{E}}}{\partial t} &
= \frac{4\pi}{c}\vec{\mathbf{j}} \\
\end{eqnarray}""")
```

Out[21]:
$$\begin{eqnarray} \nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t} & = \frac{4\pi}{c}\vec{\mathbf{j}} \\ \end{eqnarray}$$

Or you can enter latex directly with the `%%latex` cell magic:

In [22]:
```
%%latex
\begin{aligned}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\,
\frac{\partial\vec{\mathbf{E}}}{\partial t} &
= \frac{4\pi}{c}\vec{\mathbf{j}} \\
\end{aligned}
```

$$\begin{aligned} \nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t} & = \frac{4\pi}{c}\vec{\mathbf{j}} \\ \end{aligned}$$

Loading external codes

- Drag and drop a `.py` in the dashboard
- Use `%load` with any local or remote url:
  the Matplotlib Gallery!

In this notebook we've kept the output saved so you can see the result, but you should run the next cell yourself (with an active internet connection).

Let's make sure we have pylab again, in case we have restarted the kernel due to the crash demo above

In [23]:
```
%pylab inline
```

```
Welcome to pylab, a matplotlib-based Python environment
[backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [24]:
```
%load
http://matplotlib.sourceforge.net/mpl_example
s/pylab_examples/integral_demo.py
```

IPython rocks!

Just my little contribution... I have a lot of work to do but this is an exciting beginning!

You can check **here** for more information about this PR.

And you can find me at:

- **@damian_avila**
- **OQUANTA**
- **BLOG**

//