This material is included with permission from IPython documentation available online from: https://github.com/ipython/ipython-in-depth

1 A brief tour of the IPython notebook

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:

```
pwd
```

```
u'/Users/minrk/dev/ip/mine/docs/examples/notebooks'
```

```
ls
```

```
00_notebook_tour.ipynb           callbacks.ipynb                python-logo.svg
01_notebook_introduction.ipynb   cython_extension.ipynb         rmagic_extension.ipynb
Animations_and_Progress.ipynb    display_protocol.ipynb         sympy.ipynb
Capturing Output.ipynb           formatting.ipynb               sympy_quantum_computing.ipynb
Script Magics.ipynb              octavemagic_extension.ipynb    trapezoid_rule.ipynb
animation.m4v                    progbar.ipynb
```

```
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!
```

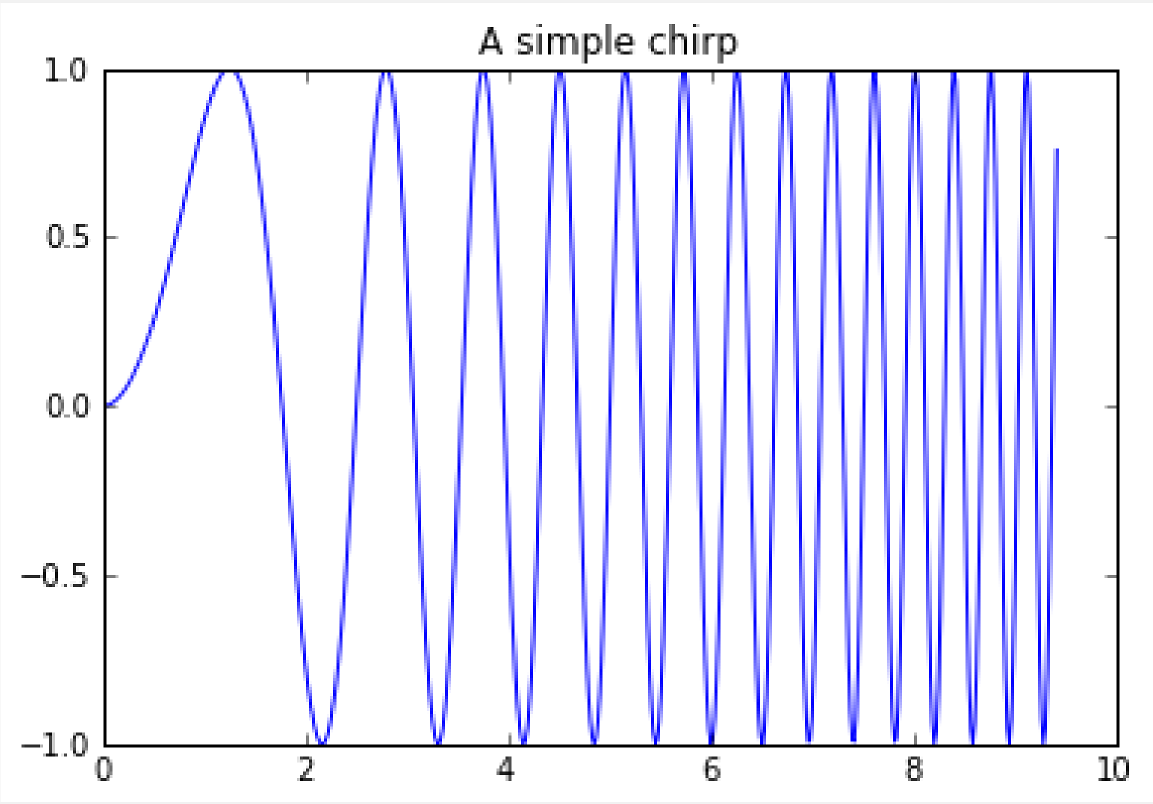
1.1 Plots with matplotlib

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

```
%pylab inline
```

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

```
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

```
>>> 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:

```
%run non_existent_file
```

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

```
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:

```
magic
```

1.2 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.

```python
import time, sys
for i in range(8):
    print i,
    time.sleep(0.5)
```

```
0

1

2

3

4

5

6

7
```

1.3 Clean crash and restart

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

```python
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!!
```

1.4 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:

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

1.5 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.

1.5.1 Images

```
from IPython.display import Image
Image(filename='../../source/_static/logo.png')
```

```
<IPython.core.display.Image at 0x10faeafd0>
```

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

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

```
<IPython.core.display.Image at 0x1060e7410>
```

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

```
from IPython.display import SVG
SVG(filename='python-logo.svg')
```

```
<IPython.core.display.SVG at 0x10fb998d0>
```

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:

```
# 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/view/images/newview.jpg')

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

Today's image from a webcam at Berkeley, (at the time I created this notebook). This should also work in the Qtconsole. Drawback is that the saved notebook will be larger, but the image will still be present offline.

```
Embed
```

```
<IPython.core.display.Image at 0x10fb99b50>
```

Today's image from same webcam at Berkeley, (refreshed every minutes, if you reload the notebook), visible only with an active internet connexion, that should be different from the previous one. This will not work on Qtconsole. Notebook saved with this kind of image will be lighter and always reflect the current version of the source, but the image won't display offline.

```
SoftLinked
```

```
<IPython.core.display.Image at 0x10fb99b10>
```

Of course, if you re-run the all notebook, the two images will be the same again.

1.5.2 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):

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

```
<IPython.lib.display.YouTubeVideo at 0x10fba2190>
```

Using the nascent video capabilities of modern browsers, you may also be able to display local videos. At the moment this doesn't work very well in all browsers, so it may or may not work for you; we will continue testing this and looking for ways to make it more robust.

The following cell loads a local file called `animation.m4v`, encodes the raw video as base64 for http transport, and uses the HTML5 video tag to load it. On Chrome 15 it works correctly, displaying a control bar at the bottom with a play/pause button and a location slider.

```
from IPython.display import HTML
video = open("animation.m4v", "rb").read()
video_encoded = video.encode("base64")
video_tag = '<video controls alt="test" src="data:video/x-m4v;base64,{0}">'.format(
    video_encoded)
HTML(data=video_tag)
```

```
<IPython.core.display.HTML at 0x10fba28d0>
```

1.6 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" />
```

and a video with the HTML5 video tag:

```
<video controls src="files/animation.m4v" />
```

These do not embed the data into the notebook file, and require that the files exist when you are viewing the notebook.

1.6.1 Security of local files

Note that this means that the IPython notebook server also acts as a generic file server for files inside the same tree as your notebooks. Access is not granted outside the notebook folder so you have strict control over what files are visible, but for this reason it is highly recommended that you do not run the notebook server with a notebook directory at a high level in your filesystem (e.g. your home directory).

When you run the notebook in a password-protected manner, local file access is restricted to authenticated users unless read-only views are active.

1.6.2 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:

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

```
<IPython.core.display.HTML at 0x1094900d0>
```

1.6.3 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:

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

d x

```
<IPython.core.display.Math at 0x10fba26d0>
```

F (k)
=
i n t $_{-\ infty}^{infty}$
f (x)
e $^{2\ pi}$
i
k

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

```
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}} \\
\nabla \cdot \vec{\mathbf{E}} & = 4 \pi \rho \\
\nabla \times \vec{\mathbf{E}}\, +\, \frac1c\, \frac{\partial\vec{\mathbf{B}}}{\partial
    t} & = \vec{\mathbf{0}} \\
\nabla \cdot \vec{\mathbf{B}} & = 0
\end{eqnarray}""")
```

b e g i n e q n a r r a y
n a b l a
t i m e s
v e c m a t h b f B
- ,
f r a c 1 c ,
f r a c p a r t i a l v e c m a t h b f E p a r t i a l
t

```
=
 f r a c 4 p i  c  v e c  m a t h b f j

 n a b l a
 c d o t
 v e c  m a t h b f E
=
4
 p i
 r h o

 n a b l a
 t i m e s
 v e c  m a t h b f E  ,
+ ,
 f r a c 1 c ,
 f r a c  p a r t i a l v e c  m a t h b f B   p a r t i a l
t
=
 v e c  m a t h b f 0

 n a b l a
 c d o t
 v e c  m a t h b f B
=
0
 e n d e q n a r r a y
<IPython.core.display.Latex at 0x10fba2c10>
```

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

```
%%latex
\begin{aligned}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t}
     & = \frac{4\pi}{c}\vec{\mathbf{j}} \\
\nabla \cdot \vec{\mathbf{E}} & = 4 \pi \rho \\
\nabla \times \vec{\mathbf{E}}\, +\, \frac1c\, \frac{\partial\vec{\mathbf{B}}}{\partial
    t} & = \vec{\mathbf{0}} \\
\nabla \cdot \vec{\mathbf{B}} & = 0
\end{aligned}
```

$$\begin{aligned}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t} & = \frac{4\pi}{c}\vec{\mathbf{j}} \\
\nabla \cdot \vec{\mathbf{E}} & = 4 \pi \rho \\
\nabla \times \vec{\mathbf{E}}\, +\, \frac1c\, \frac{\partial\vec{\mathbf{B}}}{\partial t} & = \vec{\mathbf{0}} \\
\nabla \cdot \vec{\mathbf{B}} & = 0
\end{aligned}$$

```
<IPython.core.display.Latex at 0x10a617c90>
```

There is also a `%%javascript` cell magic for running javascript directly, and `%%svg` for manually entering SVG content.

2 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

```
%pylab inline
```

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

```
%load http://matplotlib.sourceforge.net/mpl_examples/pylab_examples/integral_demo.py
```

```
#!/usr/bin/env python

# implement the example graphs/integral from pyx
from pylab import *
from matplotlib.patches import Polygon

def func(x):
    return (x-3)*(x-5)*(x-7)+85

ax = subplot(111)

a, b = 2, 9 # integral area
x = arange(0, 10, 0.01)
y = func(x)
plot(x, y, linewidth=1)

# make the shaded region
ix = arange(a, b, 0.01)
iy = func(ix)
verts = [(a,0)] + zip(ix,iy) + [(b,0)]
poly = Polygon(verts, facecolor='0.8', edgecolor='k')
ax.add_patch(poly)

text(0.5 * (a + b), 30,
     r"$\int_a^b f(x)\mathrm{d}x$", horizontalalignment='center',
     fontsize=20)
```

```
axis([0,10, 0, 180])
figtext(0.9, 0.05, 'x')
figtext(0.1, 0.9, 'y')
ax.set_xticks((a,b))
ax.set_xticklabels(('a','b'))
ax.set_yticks([])
show()
```