```
In [5]: from IPython.lib.display import YouTubeVideo
        YouTubeVideo("4vuW6tQ0218")
```

IPython Tutorial

By Paul Ivanov

Giving an IPython tutorial in front of Fernando Perez is kind of like introducing an Apple product in front of Steve Jobs.

```
In [1]: parrot = "dead"
```

```
In [2]: parrot
```

```
Out[2]: 'dead'
```

Tab completion: `obj.<tab>`

```
In [ ]: parrot.
```

Introspection:

```
In [3]: parrot.upper?
```

```
In [4]: parrot.upper()
```

```
Out[4]: 'DEAD'
```

```
In [5]: import os.path
```

Code-level introspection:

```
In [6]: os.path.join??
```

Knowing a partial name, find it

```
In [7]: # ends with "py"
        *py?
```

```
In [8]: # has "py" in it
        *py*?
```

Bring out your dead!

```
In [9]: peasant = "I'm not dead, yet"
In [10]: peasant

Out[10]: "I'm not dead, yet"
```

u'' is just a unicode string, which is what IPython uses internally , don't worry about that, you'll be stone dead

In-n-Out

You can access previous 3 results using _, __, and ___ (single-, double-, and triple-underscore).

For all others, you can use either `Out[42]` or `_42`.

Similarly, for the input strings, access the previous 3 using `_i`, `_ii`, `_iii`.

For all others, you can use either `In[42]` or `_i42`.

```
In [11]: peasant = Out[4]

In [12]: peasant

Out[12]: 'DEAD'

In [13]: peasant = peasant + _i

In [14]: peasant

Out[14]: u'DEADpeasant'

In [15]: %whos

         Variable   Type       Data/Info
         -------------------------------
         os         module     <module 'os' from '/usr/lib/python2.6/os.pyc'>
         parrot     str        dead
         peasant    unicode    DEADpeasant

In [16]: %magic

In [17]: %reset -f

In [18]: %whos

         Interactive namespace is empty.
```

Running shell commands

```
In [19]: !ipython --version

         0.13.dev
```

Getting shell output back in python

```
In [20]: v = !ipython --version
         v

Out[20]: ['0.13.dev']

In [22]: !ls

         foo  history  ipython-tutor.ipynb  talk

In [23]: files = !ls

In [24]: files?

In [25]: files.l

Out[25]: ['foo', 'history', 'ipython-tutor.ipynb', 'talk']

In [26]: files.s

Out[26]: 'foo history ipython-tutor.ipynb talk'

In [27]: files.n

Out[27]: 'foo\nhistory\nipython-tutor.ipynb\ntalk'
```

Using python variables back in the shell

```
In [28]: for x in v[0].split('.'):
             print x
             # the touch command is unix specific, just creates empty files
             !touch $x

         013dev

In [29]: ls

         0  13  dev  foo  history  ipython-tutor.ipynb  talk

In [30]: #cleanup the files we just made
         for x in v[0].split('.'):
             !rm $x

In [31]: ls

         foo  history  ipython-tutor.ipynb  talk
```

Most of the things we've covered can be found in:

```
In [32]: %quickref
In [33]: %history

         parrot = "dead"
         parrot
         parrot.upper?
         parrot.upper()
         import os.path
         os.path.join??
         # ends with "py"
         *py?
         # has "py" in it
         *py*?
         peasant = "I'm not dead, yet"
         peasant
         peasant = Out[4]
         peasant
         peasant = peasant + _i
         peasant
         %whos
         %magic
         %reset -f
         %whos
         !ipython --version
         v = !ipython --version
         v
         !ls
         !ls
         files = !ls
         files?
         files.l
         files.s
         files.n
         for x in v[0].split('.'):
             print x
             # the touch command is unix specific, just creates empty files
             !touch $x
         ls
         #cleanup the files we just made
         for x in v[0].split('.'):
             !rm $x
         ls
         %quickref
         %history
```

```
In [34]: %history -f posterity.py
```

```
In [35]: !ls

         foo  history  ipython-tutor.ipynb  posterity.py  talk
```

And finally, welcome to python!

```
In [36]: import this
In [36]: import this

         The Zen of Python, by Tim Peters

         Beautiful is better than ugly.
         Explicit is better than implicit.
         Simple is better than complex.
         Complex is better than complicated.
         Flat is better than nested.
         Sparse is better than dense.
         Readability counts.
         Special cases aren't special enough to break the rules.
         Although practicality beats purity.
         Errors should never pass silently.
         Unless explicitly silenced.
         In the face of ambiguity, refuse the temptation to guess.
         There should be one-- and preferably only one --obvious way to do it.
         Although that way may not be obvious at first unless you're Dutch.
         Now is better than never.
         Although never is often better than *right* now.
         If the implementation is hard to explain, it's a bad idea.
         If the implementation is easy to explain, it may be a good idea.
         Namespaces are one honking great idea -- let's do more of those!

In [37]: !rm posterity.py

In [ ]:
```