IPython Interactive Python shell

START HERE

obj?
Help on an object

obj.<*TAB*>
Complete attributes

%run <*script.py*>
Run a script here

%timeit <*expr*>
Time an expression

!<*command*>
Run a shell command

%debug
Debug last error

%history
Show past input

%load_ext autoreload
Reload edited modules

In[n] your input → runs → **Out[n]** result → saved as → **_ __ ___** last 3 outputs

CONCEPTS

Magics start with % (line) or %% (cell). With automagic on, the % may be left off.

Install with pip install ipython and start it with ipython.

HELP AND INSPECTION

<*obj*>**?**
Show docs for an object

<*obj*>**??**
Show source code too

<*obj*>**.**<*TAB*>
Complete attributes and methods

***<*pattern*>*?**
Search names by wildcard
*For example np.*load*? lists matching names in np.*

%psource <*obj*>
Print an object's source

%pdoc <*obj*>
Print an object's docstring

%pfile <*obj*>
Show the file defining it

%quickref
Quick reference card

%magic
Full magic system docs

%lsmagic
List all magics

HELP AND INSPECTION (cont.)

%<*magic*>**?**
Docs for one magic

RUNNING CODE

%run <*script.py*> <*args*>
Run a script in this session

%run -i <*script.py*>
Run in the session namespace

%run -t <*script.py*>
Time the script

%run -p <*script.py*>
Profile the script

%run -d <*script.py*>
Step through under pdb

%run -m <*module*>
Run a module like python -m

%load <*file*>
Load code into the prompt

%edit
Compose input in your editor

%paste
Paste clipboard code as-is

%cpaste
Paste block, end with --

%%writefile <*file*>
Write the cell to a file

%save <*file*> <*range*>
Save history lines to a file

%reset
Clear all variables

RUNNING CODE (cont.)

%reset -f
Clear without asking

TIMING

%time <*stmt*>
Time one run

%timeit <*stmt*>
Time many runs, average

%timeit -n <*N*> **-r** <*R*> <*stmt*>
Set loops and repeats

%%time
Time the whole cell

%%timeit
Time a cell, with setup line

%prun <*stmt*>
Profile a statement

HISTORY AND OUTPUT

%history
Show session input

%history -n
Include line numbers

%history -g <*text*>
Search all stored history

%history -l <*N*>
Show last N lines

%rerun
Re-run the previous line

%rerun -g <*text*>
Re-run latest match

HISTORY AND OUTPUT (cont.)

In[<*n*>**]**
Input of cell n

Out[<*n*>**]**
Output of cell n

_ / __ / ___
Last three outputs

SHELL AND SYSTEM

!<*command*>
Run a shell command

<*var*> **= !**<*command*>
Capture output as a list

!<*command*> **{**<*pyvar*>**}**
Insert a Python value
$var works too; escape a literal dollar with $$.

%%bash
Run the cell in bash

%cd <*dir*>
Change directory

%pwd
Print working directory

%env <*VAR*>**=**<*value*>
Set an env variable

%pip install <*pkg*>
Install into this environment

DEBUGGING

%debug
Open pdb after an exception

DEBUGGING (cont.)

%pdb on
Debug every uncaught error

%xmode Verbose
Richer tracebacks
Modes: Plain, Context, Verbose, Minimal.

%tb
Show last traceback again

MODULES AND WORKSPACE

%load_ext autoreload
Enable the extension

%autoreload 2
Reload all before each run

%autoreload 1
Reload only %aimport modules

%who
List variable names

%whos
List variables with types

%matplotlib inline
Plot inside the frontend

CONFIG AND SHORTCUTS

ipython profile create
Create default config files

ipython --profile=<*name*>
Start with a profile

ipython -i <*script.py*>
Run, then stay interactive

CONFIG AND SHORTCUTS (cont.)

Keyboard

Key	Does
Tab	Complete or cycle
Ctrl+R	Search history
Ctrl+O	Open new line
Esc Enter	Force run block
Ctrl+L	Clear screen
Ctrl+D	Exit

Startup scripts: put .py or .ipy files in ~/.ipython/profile_default/startup/; they run in filename order.

RECIPES

Edit a module and keep going

```
%load_ext autoreload
%autoreload 2
import mymod
mymod.run()
# edit mymod.py, then call again
```

Debug a failing script

```
%run script.py
%debug
# or start under pdb from the top:
%run -d script.py
```

Compare two implementations

```
%timeit sorted(data)
%timeit -n 100 -r 5 list_sort(data)
```

Shell output into Python

```
files = !ls *.csv
for f in files: print(f)
%%bash
grep -c error app.log
```