

IPython adds on top of the normal Python syntax. These are known in IPython as *magic commands*, and are prefixed by the % character. These magic commands are designed to succinctly solve various common problems in standard data analysis. Magic commands come in two flavors: *line magics*, which are denoted by a single % prefix and operate on a single line of input, and *cell magics*, which are denoted by a double %% prefix and operate on multiple lines of input. We'll demonstrate and discuss a few brief examples here, and come back to more focused discussion of several useful magic commands later in the chapter.

Pasting Code Blocks: %paste and %cpaste

When you're working in the IPython interpreter, one common gotcha is that pasting multiline code blocks can lead to unexpected errors, especially when indentation and interpreter markers are involved. A common case is that you find some example code on a website and want to paste it into your interpreter. Consider the following simple function:

```
>>> def donothing(x):
...     return x
```

The code is formatted as it would appear in the Python interpreter, and if you copy and paste this directly into IPython you get an error:

```
In [2]: >>> def donothing(x):
...:         ...     return x
...:
...:
File "<ipython-input-20-5a66c8964687>", line 2
...     return x
          ^
SyntaxError: invalid syntax
```

In the direct paste, the interpreter is confused by the additional prompt characters. But never fear—IPython's %paste magic function is designed to handle this exact type of multiline, marked-up input:

```
In [3]: %paste
>>> def donothing(x):
...     return x

## -- End pasted text --
```

The %paste command both enters and executes the code, so now the function is ready to be used:

```
In [4]: donothing(10)
Out[4]: 10
```

A command with a similar intent is %cpaste, which opens up an interactive multiline prompt in which you can paste one or more chunks of code to be executed in a batch:

```
In [5]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:>>> def donothing(x):
:...     return x
:--
```

These magic commands, like others we'll see, make available functionality that would be difficult or impossible in a standard Python interpreter.

Running External Code: %run

As you begin developing more extensive code, you will likely find yourself working in both IPython for interactive exploration, as well as a text editor to store code that you want to reuse. Rather than running this code in a new window, it can be convenient to run it within your IPython session. This can be done with the %run magic.

For example, imagine you've created a *myscript.py* file with the following contents:

```
#-----
# file: myscript.py

def square(x):
    """square a number"""
    return x ** 2

for N in range(1, 4):
    print(N, "squared is", square(N))
```

You can execute this from your IPython session as follows:

```
In [6]: %run myscript.py
1 squared is 1
2 squared is 4
3 squared is 9
```

Note also that after you've run this script, any functions defined within it are available for use in your IPython session:

```
In [7]: square(5)
Out[7]: 25
```

There are several options to fine-tune how your code is run; you can see the documentation in the normal way, by typing **%run?** in the IPython interpreter.

Timing Code Execution: %timeit

Another example of a useful magic function is %timeit, which will automatically determine the execution time of the single-line Python statement that follows it. For example, we may want to check the performance of a list comprehension:

```
In [8]: %timeit L = [n ** 2 for n in range(1000)]
1000 loops, best of 3: 325 µs per loop
```