```
In [1]:
(reverse-i-search)`sqa': square??
```

At any point, you can add more characters to refine the search, or press Ctrl-r again to search further for another command that matches the query. If you followed along in the previous section, pressing Ctrl-r twice more gives:

```
In [1]:
(reverse-i-search)`sqa': def square(a):
    """Return the square of a"""
    return a ** 2
```

Once you have found the command you're looking for, press Return and the search will end. We can then use the retrieved command, and carry on with our session:

```
In [1]: def square(a):
    """Return the square of a"""
    return a ** 2

In [2]: square(2)
Out[2]: 4
```

Note that you can also use Ctrl-p/Ctrl-n or the up/down arrow keys to search through history, but only by matching characters at the beginning of the line. That is, if you type **`def`** and then press Ctrl-p, it would find the most recent command (if any) in your history that begins with the characters `def`.

Miscellaneous Shortcuts

Finally, there are a few miscellaneous shortcuts that don't fit into any of the preceding categories, but are nevertheless useful to know:

Keystroke	Action
Ctrl-l	Clear terminal screen
Ctrl-c	Interrupt current Python command
Ctrl-d	Exit IPython session

The Ctrl-c shortcut in particular can be useful when you inadvertently start a very long-running job.

While some of the shortcuts discussed here may seem a bit tedious at first, they quickly become automatic with practice. Once you develop that muscle memory, I suspect you will even find yourself wishing they were available in other contexts.

IPython Magic Commands

The previous two sections showed how IPython lets you use and explore Python efficiently and interactively. Here we'll begin discussing some of the enhancements that

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: