

01.04-Input-Output-History

February 27, 2017

This notebook contains an excerpt from the [Python Data Science Handbook](#) by Jake VanderPlas; the content is available [on GitHub](#).

The text is released under the [CC-BY-NC-ND license](#), and code is released under the [MIT license](#). If you find this content useful, please consider supporting the work by [buying the book](#)!

< [IPython Magic Commands](#) | [Contents](#) | [IPython and Shell Commands](#) >

1 Input and Output History

Previously we saw that the IPython shell allows you to access previous commands with the up and down arrow keys, or equivalently the Ctrl-p/Ctrl-n shortcuts. Additionally, in both the shell and the notebook, IPython exposes several ways to obtain the output of previous commands, as well as string versions of the commands themselves. We'll explore those here.

1.1 IPython's In and Out Objects

By now I imagine you're quite familiar with the `In [1]:/Out[1]:` style prompts used by IPython. But it turns out that these are not just pretty decoration: they give a clue as to how you can access previous inputs and outputs in your current session. Imagine you start a session that looks like this:

```
In [1]: import math
```

```
In [2]: math.sin(2)
Out[2]: 0.9092974268256817
```

```
In [3]: math.cos(2)
Out[3]: -0.4161468365471424
```

We've imported the built-in `math` package, then computed the sine and the cosine of the number 2. These inputs and outputs are displayed in the shell with In/Out labels, but there's more—IPython actually creates some Python variables called `In` and `Out` that are automatically updated to reflect this history:

```
In [4]: print(In)
['', 'import math', 'math.sin(2)', 'math.cos(2)', 'print(In)']
```

```
In [5]: Out
Out[5]: {2: 0.9092974268256817, 3: -0.4161468365471424}
```

The `In` object is a list, which keeps track of the commands in order (the first item in the list is a place-holder so that `In[1]` can refer to the first command):

```
In [6]: print(In[1])
import math
```

The `Out` object is not a list but a dictionary mapping input numbers to their outputs (if any):

```
In [7]: print(Out[2])
0.9092974268256817
```

Note that not all operations have outputs: for example, `import` statements and `print` statements don't affect the output. The latter may be surprising, but makes sense if you consider that `print` is a function that returns `None`; for brevity, any command that returns `None` is not added to `Out`.

Where this can be useful is if you want to interact with past results. For example, let's check the sum of `sin(2) ** 2` and `cos(2) ** 2` using the previously-computed results:

```
In [8]: Out[2] ** 2 + Out[3] ** 2
Out[8]: 1.0
```

The result is `1.0` as we'd expect from the well-known trigonometric identity. In this case, using these previous results probably is not necessary, but it can become very handy if you execute a very expensive computation and want to reuse the result!

1.2 Underscore Shortcuts and Previous Outputs

The standard Python shell contains just one simple shortcut for accessing previous output; the variable `_` (i.e., a single underscore) is kept updated with the previous output; this works in IPython as well:

```
In [9]: print(_)
1.0
```

But IPython takes this a bit further—you can use a double underscore to access the second-to-last output, and a triple underscore to access the third-to-last output (skipping any commands with no output):

```
In [10]: print(__)
-0.4161468365471424
```

```
In [11]: print(____)
0.9092974268256817
```

IPython stops there: more than three underscores starts to get a bit hard to count, and at that point it's easier to refer to the output by line number.

There is one more shortcut we should mention, however—a shorthand for `Out[X]` is `_X` (i.e., a single underscore followed by the line number):

```
In [12]: Out[2]
Out[12]: 0.9092974268256817
```

```
In [13]: _2
Out[13]: 0.9092974268256817
```

1.3 Suppressing Output

Sometimes you might wish to suppress the output of a statement (this is perhaps most common with the plotting commands that we'll explore in [Introduction to Matplotlib](#)). Or maybe the command you're executing produces a result that you'd prefer not like to store in your output history, perhaps so that it can be deallocated when other references are removed. The easiest way to suppress the output of a command is to add a semicolon to the end of the line:

```
In [14]: math.sin(2) + math.cos(2);
```

Note that the result is computed silently, and the output is neither displayed on the screen or stored in the Out dictionary:

```
In [15]: 14 in Out
Out[15]: False
```

1.4 Related Magic Commands

For accessing a batch of previous inputs at once, the `%history` magic command is very helpful. Here is how you can print the first four inputs:

```
In [16]: %history -n 1-4
1: import math
2: math.sin(2)
3: math.cos(2)
4: print(In)
```

As usual, you can type `%history?` for more information and a description of options available. Other similar magic commands are `%rerun` (which will re-execute some portion of the command history) and `%save` (which saves some set of the command history to a file). For more information, I suggest exploring these using the `? help` functionality discussed in [Help and Documentation in IPython](#).

< [IPython Magic Commands](#) | [Contents](#) | [IPython and Shell Commands](#) >