

The curly braces contain the variable name, which is replaced by the variable's contents in the shell command.

## Shell-Related Magic Commands

If you play with IPython's shell commands for a while, you might notice that you cannot use `!cd` to navigate the filesystem:

```
In [11]: !pwd  
/home/jake/projects/myproject
```

```
In [12]: !cd ..
```

```
In [13]: !pwd  
/home/jake/projects/myproject
```

The reason is that shell commands in the notebook are executed in a temporary sub-shell. If you'd like to change the working directory in a more enduring way, you can use the `%cd` magic command:

```
In [14]: %cd ..  
/home/jake/projects
```

In fact, by default you can even use this without the `%` sign:

```
In [15]: cd myproject  
/home/jake/projects/myproject
```

This is known as an automagic function, and this behavior can be toggled with the `%automagic` magic function.

Besides `%cd`, other available shell-like magic functions are `%cat`, `%cp`, `%env`, `%ls`, `%man`, `%mkdir`, `%more`, `%mv`, `%pwd`, `%rm`, and `%rmdir`, any of which can be used without the `%` sign if automagic is on. This makes it so that you can almost treat the IPython prompt as if it's a normal shell:

```
In [16]: mkdir tmp
```

```
In [17]: ls  
myproject.txt tmp/
```

```
In [18]: cp myproject.txt tmp/
```

```
In [19]: ls tmp  
myproject.txt
```

```
In [20]: rm -r tmp
```

This access to the shell from within the same terminal window as your Python session means that there is a lot less switching back and forth between interpreter and shell as you write your Python code.

# Errors and Debugging

Code development and data analysis always require a bit of trial and error, and IPython contains tools to streamline this process. This section will briefly cover some options for controlling Python's exception reporting, followed by exploring tools for debugging errors in code.

## Controlling Exceptions: %xmode

Most of the time when a Python script fails, it will raise an exception. When the interpreter hits one of these exceptions, information about the cause of the error can be found in the *traceback*, which can be accessed from within Python. With the %xmode magic function, IPython allows you to control the amount of information printed when the exception is raised. Consider the following code:

```
In[1]: def func1(a, b):
        return a / b

        def func2(x):
            a = x
            b = x - 1
            return func1(a, b)

In[2]: func2(1)

-----
ZeroDivisionError                                Traceback (most recent call last)

<ipython-input-2-b2e110f6fc8f> in <module>()
----> 1 func2(1)

<ipython-input-1-d849e34d61fb> in func2(x)
      5     a = x
      6     b = x - 1
----> 7     return func1(a, b)

<ipython-input-1-d849e34d61fb> in func1(a, b)
      1 def func1(a, b):
----> 2     return a / b
      3
      4 def func2(x):
      5     a = x

ZeroDivisionError: division by zero
```

Calling func2 results in an error, and reading the printed trace lets us see exactly what happened. By default, this trace includes several lines showing the context of each