

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

step that led to the error. Using the `%xmode` magic function (short for *exception mode*), we can change what information is printed.

`%xmode` takes a single argument, the mode, and there are three possibilities: `Plain`, `Context`, and `Verbose`. The default is `Context`, and gives output like that just shown. `Plain` is more compact and gives less information:

```
In[3]: %xmode Plain
Exception reporting mode: Plain

In[4]: func2(1)
-----
Traceback (most recent call last):

File "<ipython-input-4-b2e110f6fc8f>", line 1, in <module>
    func2(1)

File "<ipython-input-1-d849e34d61fb>", line 7, in func2
    return func1(a, b)

File "<ipython-input-1-d849e34d61fb>", line 2, in func1
    return a / b
```

ZeroDivisionError: division by zero

The `Verbose` mode adds some extra information, including the arguments to any functions that are called:

```
In[5]: %xmode Verbose
Exception reporting mode: Verbose

In[6]: func2(1)
-----
ZeroDivisionError                                Traceback (most recent call last)

<ipython-input-6-b2e110f6fc8f> in <module>()
----> 1 func2(1)
      global func2 = <function func2 at 0x103729320>

<ipython-input-1-d849e34d61fb> in func2(x=1)
      5     a = x
      6     b = x - 1
----> 7     return func1(a, b)
      global func1 = <function func1 at 0x1037294d0>
      a = 1
      b = 0
```

```

<ipython-input-1-d849e34d61fb> in func1(a=1, b=0)
      1 def func1(a, b):
----> 2     return a / b
      a = 1
      b = 0
      3
      4 def func2(x):
      5     a = x

```

```
ZeroDivisionError: division by zero
```

This extra information can help you narrow in on why the exception is being raised. So why not use the Verbose mode all the time? As code gets complicated, this kind of traceback can get extremely long. Depending on the context, sometimes the brevity of Default mode is easier to work with.

Debugging: When Reading Tracebacks Is Not Enough

The standard Python tool for interactive debugging is `pdb`, the Python debugger. This debugger lets the user step through the code line by line in order to see what might be causing a more difficult error. The IPython-enhanced version of this is `ipdb`, the IPython debugger.

There are many ways to launch and use both these debuggers; we won't cover them fully here. Refer to the online documentation of these two utilities to learn more.

In IPython, perhaps the most convenient interface to debugging is the `%debug` magic command. If you call it after hitting an exception, it will automatically open an interactive debugging prompt at the point of the exception. The `ipdb` prompt lets you explore the current state of the stack, explore the available variables, and even run Python commands!

Let's look at the most recent exception, then do some basic tasks—print the values of `a` and `b`, and type **quit** to quit the debugging session:

```

In[7]: %debug
> <ipython-input-1-d849e34d61fb>(2)func1()
      1 def func1(a, b):
----> 2     return a / b
      3

ipdb> print(a)
1
ipdb> print(b)
0
ipdb> quit

```