

Re: Having to "print" before method invocation?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-03/msg01371.html>

- *From:* "Jeremy L. Moles" <jeremy@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 08 Mar 2006 10:17:08 -0500
-

Hey Fredrik, thanks for responding. :) Your posts are always helpful and informative!

On Wed, 2006-03-08 at 15:41 +0100, Fredrik Lundh wrote:

Jeremy L. Moles wrote:

I have an object (written as part C extension, part pure Python) called foo that I've been using without much fuss for a few months now. However, in my latest project (a rather large one involving multi-threading, pygtk, etc.), I'm seeing some really strange behavior with a particular instance of my foo object.

About midway through my program, any attempt to use the instance fails; however, if I add print statements before trying to invoke methods on it, the foo object instance works fine.

fails in what way?

Unfortunately, very silently. There aren't any exceptions thrown or anything. The be more specific, the C part of the object is a wrapper around some socket Send/Recv functions (that interface w/ wpa_supplicant). The problem I'm getting is that unless I add that print statement, wpaobject.Recv() returns "", whereas with the print statement it returns what it should (a large string representing the response from wpa_supplicant).

More strange is that this is the first time I've had to do this. Up until this point, every unittest and app has succeeded without such trickery.

if you get a spurious exception, it's very likely that your C extension sets the exception state (either directly or because some API function it uses fails), but forgets to report this back to Python.

Re: Having to "print" before method invocation?

e.g. if you have a C function that does something like

```
PyErr_SetString(PyExc_AttributeError, "blah blah");
```

```
Py_INCREF(Py_None);  
return Py_None;
```

instead of

```
PyErr_SetString(PyExc_AttributeError, "blah blah");
```

```
return NULL;
```

the interpreter won't raise the exception immediately (since it expected you to return NULL if something went wrong), but the exception may still be raised at a later time, if you run interpreter code that does something like

```
do something  
if (PyErr_Occurred())  
... /* this will catch your error even if "something" succeeds */ ...
```

or it may be masked, by code that does

```
PyErr_Clear();  
do something
```

the actual exception might give you additional clues (e.g. if you get a `KeyError`, look for unchecked dictionary accesses in your code, etc).

hope this helps!

</F>