

Re: from __future__ import absolute_import ?

Re: from __future__ import absolute_import ?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-02/msg00466.html>

- *From:* Ron Adam <rrr@xxxxxxxxxxxx>
 - *Date:* Sat, 03 Feb 2007 10:30:35 -0600
-

Peter Otten wrote:

Ron Adam wrote:

```
work
|
|-- foo.py # print "foo not in bar"
|
|-- bar
|
|-- __init__.py
|
|-- foo.py # print "foo in bar"
|
|-- absolute.py # from __futer__ import absolute_import
| # import foo
|
|-- relative.py # import foo
```

* Where "work" is in the path.

(1)

```
C:\work>python -c "import bar.absolute"
foo not in bar
```

```
C:\work>python -c "import bar.relative"
foo in bar
```

(2)

```
C:\work>python -m "bar.absolute"
foo not in bar
```

Re: from __future__ import absolute_import ?

```
C:\work>python -m "bar.relative"  
foo not in bar
```

(3)

```
C:\work>python  
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)]  
on win 32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import bar.absolute  
foo not in bar  
>>> import bar.relative  
foo in bar
```

(4)

```
C:\work>cd bar
```

A path below the package level is generally a good means to shoot yourself in the foot and should be avoided with or without absolute import.

Seems so. :-/

```
C:\work\bar>python -c "import bar.absolute"  
foo in bar
```

```
C:\work\bar>python -c "import bar.relative"  
foo in bar
```

(5)

```
C:\work\bar>python  
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)]  
on win 32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import bar.absolute  
foo in bar  
>>> import bar.relative  
foo in bar  
>>>
```

Case (2) seems like it is a bug.

Re: from __future__ import absolute_import ?

Re: from __future__ import absolute_import ?

I think so, too.

This one is the reasons I had trouble figuring it out. I was using the `-m` command option when I tried to test it.

There is a bug report on absolute/relative imports already. I'm not sure if this particular item is covered under it or not. Doesn't sound like it as the bug report address the relative aspects of it.

Why not also have (4), and (5) do the same as cases (1) and (3)?

The `work/bar` directory is the current working directory and occurs in the path before the `work` directory.

Yes. Unfortunately this is a side effect of using the os's directory structure to represent a python "package" structure. If a package was represented as a combined single file. Then the working directory would always be the package directory.

> When `bar.absolute` imports `foo` python is
> unaware that `work/bar/foo.py` is part of the `bar` package.

Umm.... isn't the "bar" stuck on the front of "bar.absolute" a pretty obvious hint. ;-)

If you run the module directly as a file...

```
python bar/foo.py  
or python foo.py
```

Then I can see that it doesn't know. But even then, it's possible to find out. ie... just check for an `__init__.py` file.

Python has a certain minimalist quality where it tries to do a lot with a minimum amount of resources, which I generally love. But in this situation that might not be the best thing. It would not be difficult for python to detect if a module is in a package, and determine the package location. With the move to explicit absolute/relative imports, it would make sense if python also were a little smarter in this area.

in cases (4) and (5), that is the result I would expect if I did:

```
import absolute # with no 'bar.' prefix.  
import relative
```

From what I understand, in 2.6 relative imports will be depreciated, and in 2.7

Re: from __future__ import absolute_import ?

Re: from __future__ import absolute_import ?

they will raise an error. (providing plans don't change)

Would that mean the absolute imports in (4) and (5) would either find the 'foo not in bar' or raise an error?

No, in 1, 3 --- and 2 if the current behaviour is indeed a bug. This is only for the relative import which would have to be spelt

```
from . import foo
```

Was that a 'yes' for examples 4 and 5, since 1,2 and 3 are 'no'?

in an absolute-import-as-default environment;

```
import foo
```

would always be an absolute import.

But what is the precise meaning of "absolute import" in this un-dotted case?

Currently it is:

"A module or package that is located in sys.path or the current directory".

But maybe a narrower interpretation may be better?:

"A module or package found in sys.path, or the current directory and is **not** in a package."

If it's in a package then the dotted "absolute" name should be used. Right?

I guess what I'm getting at, is it would be nice if the following were always true.

```
from __import__ import absolute_import
```

```
import thispackage.module
import thispackage.subpackage
```

```
# If thispackage is the same name as the current package,
# then do not look on sys.path.
```

```
import otherpackage.module
import otherpackage.subpackage
```

Re: from __future__ import absolute_import ?

Re: from __future__ import absolute_import ?

```
# If otherpackage is a different name from the current package,  
# then do not look in this package.
```

```
import module  
import package
```

```
# Module and package are not in a package, even the current one,  
# so don't look in any packages, even if the current directory is  
# in this (or other) package.
```

If these were always true, :-) I think it avoid some situations where things don't work, or don't work like one would expect.

In addition to the above, when executing modules directly from a directory inside a package, if python were to detect the package and then follow these same rules. It would avoid even more surprises. While you are editing modules in a package, you could then run them directly and get the same behavior you get if you cd'd out of the package and then ran it.

All in all, what I'm suggesting is that the concept of a package (type) be much stronger than that of a search path or current directory. And that this would add a fair amount of reliability to the language.

IMHO, of course. :-)

Cheers,
Ron

If so, is there any way to force (warning/error) behavior now?

I don't know.

Peter

Re: from __future__ import absolute_import ?