Module and Package

June 26, 2016

Contents

Modules

- Any Python source file is a module

```
# spam.py
def grok(x):
    ...

def blah(x):
    ...
```

- You use import to execute and access it

```
a = spam.grok('hello')

from spam import grok
a = grok('hello')
```

Namespaces

- Each module is its own isolated world

```
# spam.py

x = 42

def blah():
    print(x)

# eggs.py

x = 37

def foo():
    print(x)
```

- These definitions of x are different
- What happens in Module, stays in a module

Global Variables

- Global variables bind inside the same module

```
# spam.py

x = 42

def blah():
    print(X)
```

- Fuctions record their definition environment

```
>>> from spam import blah
>>> blah.__module__
'spam'
>>> blah.__globals__
{ 'x': 42, ...}
>>>
```

Module Execution

- When a module is imported, `all of the statements in the module execute` one after another until the end of the file is reached
- The contents of the module namespace are of the global names that are still defined at the end of the execution process
- If there are scripting statements that carry out tasks in the global scope (printing, creating files, etc.), yout will see them run on import

`from module import`

- Lifts selected symbols out of a module **after importing it** and makes them available locally

```
from math import sin, cos

def rectangular(r, theta):
    x = r * cos(theta)
    y = r * sin(theta)
    return x, y
```

- Allows parts of module to be used without having to type the module prefix

`from module import *`

- Takes **all symbols** from a module and places them into local scope

```
from math import *

def rectangular(r, theta):
    y = x * cos(theta)
    y = r * sin(theta)
    return x, y
```

- Sometimes useful
- Usually considered bad style (try to avoid)

Commentary

- Variations on import do not change the way that modules work

```
import math as m
from math import cos, sin
from math import *
...
```

- import always executes the **entire** file
- Modules are still isolated environments
- These variations are just manipulating names

Module Names

- File names have to follow the rules

```
# good.py

...

# 2bad.py

...
```

- Comment: This mistake comes up a lot when teaching Python to newcomers
- Must be a valid identifier name
- Also: avoid non-ASCII characters

Naming Conventions

- It is standard practice for package and module names to be concise and lowercase
- `foo.py`
- **not** `MyFooModule.py`

Module Search Path

- If a file isn't on the path, it won't import

```
>>> import sys
>>> sys.path ['',
      '/usr/local/lib/python34.zip',
      '/usr/local/lib/python3.4',
      '/usr/local/lib/python3.4/plat-darwin',
      '/usr/local/lib/python3.4/lib-dynload',
      '/usr/local/lib/python3.4/site-packages']
```

- Sometimes you might hack it…although doing so feels "dirty"

```
import sys
      sys.path.append("/project/foo/myfiles")
```

Module Cashe

- Modules only get loaded once
- There's a cache behind the scenes

```
>>> import spam
>>> import sys
>>> 'spam' in sys.modules
True
>>> sys.modules['spam']
<module 'spam' from 'spam.py'>
```

Consequence If you make a change to the source and repeat the import, nothing happens (often furstrating to newcomers)

Module Reloading

- You can force-reload a module, but you're never supposed to do it

```
>>> from importlib import reload
>>> reload(spam)
<module 'spam' from 'spam.py'>
```

- Apparently zombies are spawned if you do this
- No, seriously
- Don't do it

`__main__` check

- If a file might run as a main program, do this

```
# spam.py

...
if __name__ == '__main__':
    # Running as the main program
    ...
```

- Such code won't run on library import

```
import spam    # Main code doesn't execute
```

```
zsh % python spam.py  # Main code executes
```

Packages

- For larger collectons of code, it is usually desirable to organize modules into a hierarchy

```
|--spam/
   |-- foo.py
   |-- bar/
       |-- grok.py
   ...
```

- To do it, you just add init.py files

```
|--spam/
   |-- init.py
   |-- foo.py
   |-- bar/
       |-- grok.py
   ...
```

Using a Package

- import works the same way, multiple levels

```
import spam.foo
from spam.bar import grok
```

- The `__init__.py` file import at each level
- Apparently you can do things in those files
- We'll get to that

Comments

- At a simple level, there's not much to `import`
- ...except for everything else