from foo import spam

Sasi donaparthi
@sdonapar
Sasidhar.Donaparthi@gmail.com

A bit of History

- Prior to Python 3.3 importlib was using imp C library
- From Python 3.3, importlib is the default import library
- Importlib has been re-implemented in pure python
- Conceptually, there is no major chnage in the mechanics of how importlib works between python 2.5 and 3.3

Changed in version 3.3: The import system has been updated to fully implement the second phase of PEP 302. There is no longer any implicit import machinery - the full import system is exposed through `sys.meta_path`. In addition, native namespace package support has been implemented (see PEP 420).

Terminology

Python has only one type of module object, and all modules are of this type, regardless of whether the module is implemented in Python, C, or something else. To help organize modules and provide a naming hierarchy, Python has a concept of packages.

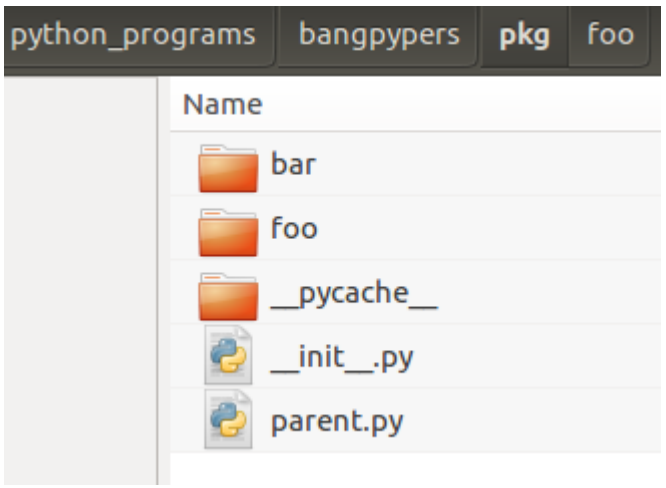
It's important to keep in mind that all packages are modules, but not all modules are packages. Or put another way, packages are just a special kind of module. Specifically, any module that contains a `__path__` attribute is considered a package.

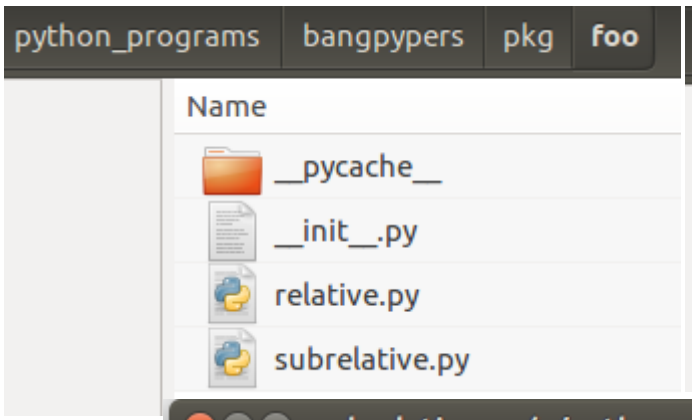
- Finder
  - Finds a module
- Loader
  - Loads a module
- Meta path finder
  - Found on sys.meta_path
  - Get you a finder
- Path entry finder
  - Searches in the entry from import path

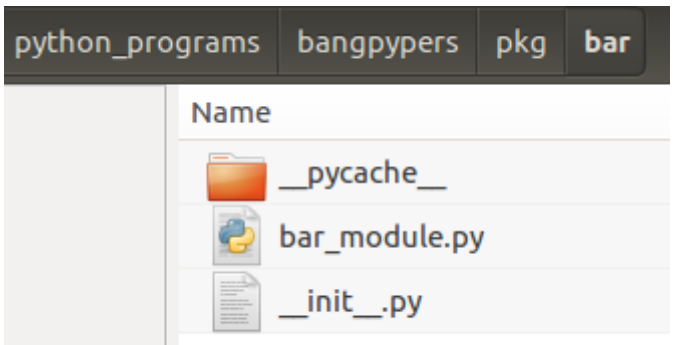
First Step

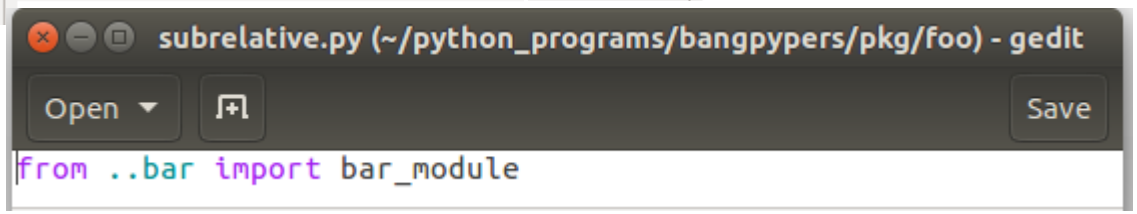
from ..bar import bar_module

__import__('bar',globals(),locals(),['bar_module'],2)









Import System

- Module Cache (sys.modules)
- Finders – sys.meta_path
  - find_module
- Path Finders
  - sys.path_importer_cache
  - sys.path_hooks
- Loader
  - Get source code
  - Compile to bytecode
  - Create module object
  - Exec the bytecode in the module dict scope

Module Cache

- All of the modules are cached in sys.modules dictionary
  - sys.modules[‘fullname’] = <module object>
- Import process checks sys.modules before it begins any other processing
- Lock is at the module level from python 3.3

Finders

- Built-in modules finder
- Frozen modules finder
- Default path finder

```
>>> import sys
>>> for finder in sys.meta_path:
...     print(finder)
...
<class '_frozen_importlib.BuiltinImporter'>
<class '_frozen_importlib.FrozenImporter'>
<class '_frozen_importlib_external.PathFinder'>
```

```
6  for finder in sys.meta_path:
7      loader = finder.find_module(name)
8      if loader:
9          return loader.load_module(name)
10
11 raise ImportError
```

namespace package - A PEP 420 package which serves only as a container for subpackages, and specifically are not like a regular package because they have no __init__.py file.

Path Finders

- Check in sys.path_importer_cache
- Then in sys.path_hooks

```
>>> import sys
>>> for my_dir in sys.path_importer_cache:
...     print(my_dir)
...
/home/sasidhar/anaconda36/lib/python36.zip
/home/sasidhar/anaconda36/lib/python3.6
/home/sasidhar/anaconda36/lib/python3.6/encodings
/home/sasidhar/anaconda36/lib/python3.6/lib-dynload
/home/sasidhar/anaconda36/lib/python3.6/site-packages
/home/sasidhar/anaconda36/lib/python3.6/site-packages/Sphinx-1.5.1-py3.6.egg
/home/sasidhar/anaconda36/lib/python3.6/site-packages/setuptools-27.2.0-py3.6.egg
/home/sasidhar/python_programs/bangpypers
```

```
>>> import sys
>>> for path_hook in sys.path_hooks:
...     print(path_hook)
...
<class 'zipimport.zipimporter'>
<function FileFinder.path_hook.<locals>.path_hook_for_FileFinder at 0x7f65d1ef26a8
>
```

Loader

- Finder will find a loader
- Module is created by calling load_module method of loader object
- Bytecode (pyc files) is written to the disk

ModuleSpec

- Module specs were introduced in Python 3.4, by PEP 451.
- A namespace containing the import-related information used to load a module. An instance of importlib.machinery.ModuleSpec.

Module Object

- Read the code from the source file
- Create a new module object
- Compile the code
- Execute the code in the scope of newly created module object

```
1  import types
2
3  def import_module(module_name):
4      sourcepath = module_name + ".py"
5      with open(sourcepath, "r") as module_file:
6          sourcecode = module_file.read()
7      mod = types.ModuleType(module_name)
8      mod.__file__ = sourcepath
9      code = compile(sourcecode, sourcepath, 'exec')
10     exec(code, mod.__dict__)
11     return mod
```

Python 3.6

```
module = None
if spec.loader is not None and hasattr(spec.loader, 'create_module'):
    # It is assumed 'exec_module' will also be defined on the loader.
    module = spec.loader.create_module(spec)
if module is None:
    module = ModuleType(spec.name)
# The import-related module attributes get set here:
_init_module_attrs(spec, module)
```

```
if spec.loader is None:
    if spec.submodule_search_locations is not None:
        # namespace package
        sys.modules[spec.name] = module
    else:
        # unsupported
        raise ImportError
elif not hasattr(spec.loader, 'exec_module'):
    module = spec.loader.load_module(spec.name)
    # Set __loader__ and __package__ if missing.
else:
    sys.modules[spec.name] = module
    try:
        spec.loader.exec_module(module)
    except BaseException:
        try:
            del sys.modules[spec.name]
        except KeyError:
            pass
        raise
return sys.modules[spec.name]
```

Find moduleSpec

Create a module object

Add module object to sys.modules

Execute module

Return module object

What I can’t do ?

- Modify start-up
- Modify __main__
- Change language features

What can you do ?

- Logging imports
- Import from remote hosts
- Virtual import paths
- Fix Circular imports
- Lazy imports
- Post import hooks
- Custom module types

References

- Python Documentation
- How Import Works - Brett Cannon
- David Beazley - Modules and Packages: Live an d Let Die!
- Getting the Most Out of Python Imports

Thank you