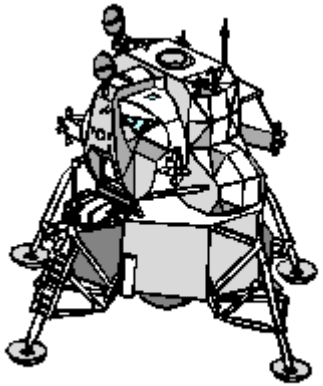
python™ 2 programming

Modules and Packages

Modules and Packages

- **Contents**
  - **What are modules and packages?**
  - **How does Python find a module?**
  - **Multiple source files**
  - **Importing a module**
  - **Importing names**
  - **Directories as packages**
  - **Writing a module**
  - **Module documentation**
  - **Distributing packages**
    - Folder , Egg , Wheel



- **Summary**
  - **Python debugger and profiler**

What are modules?

- A module is a file containing code
  - Usually, but not exclusively, written in Python
  - Usually with a .py filename suffix (some modules are built-in)
  - Contains functions, which may be exported
- A module might be byte-code
  - Python will create a .pyc file if none exists
  - Python will overwrite this if the .py file is younger
  - This gives a faster load-time at program start-up
- A module might be a DLL or shared object
  - With a .pyd filename suffix
  - Often written in C as a Python extension

What are packages?

- **A package is a logical group of modules**
- **A directory containing a set of modules is a package**
- **The difference is a file called `__init__.py`**
  - **Often empty**
  - **Can contain initialisation code**
  - **Can even contain functions**
  - **Can contain a list of the public interfaces as attribute `__all__`**
    - These are the names imported with `from` *Module* `import *`

```
# Public interface
__all__ = ['GetProcs', 'GetProcsAll', 'filter']
```

Multiple source files

- **Why bother?**
  - **Increase maintainability**
    - Independent modules can be understood easily
  - **Functional decomposition**
    - Simplify the implementation
  - **Encapsulation & information hiding**
    - Easier re-use of modules in a different program
    - Easier to change module without affecting the entire program
  - **Support concurrent development**
    - Multiple people working simultaneously
    - Debug separately in discrete units
  - **Promote reuse**
    - Logical variable and function names can safely be reused
    - Use or adapt available standard modules

How does Python find a module?

- **The initial path is from sys.path**
  - **May be modified using `sys.path.append(`*dirname*`)`**
  - **The first directory is the directory from which the main program was loaded**

```
import sys
sys.path.append('./DemoModules')
import mymodule
print sys.path
```

```
['C:\\QA\\Python\\MyDemos', 'C:\\Python26\\Lib', ...
./DemoModules]
```

- **Or change environment variable PYTHONPATH**
  - **Contains a list of directories to be searched**
  - **Separator is the same as your system's PATH**
    - **:** for *NIX ; for Windows

Importing a module

- **Surprisingly, use the import command**
  - **At the top of your program, by convention**

```
import MyModule
print MyModule.attribute
```

Case sensitive, even on Windows

  - **Can specify a comma-separated list of module names**

```
import MyModuleA, MyModuleB, MyModuleC
```

  - **Can specify an alias for a module name**

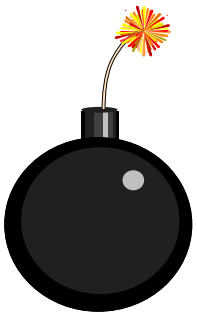
```
import MyModuleWin32 as MyModule
print MyModule.attribute
```

  - **Trouble is, you have to specify the module name for each call**

Importing names

- **Alternatively, import the names into your namespace**

```
from MyModule import *
```



  - **Beware! Risk of name collisions!**

- **Specify specific object name(s)**

```
from MyModule import MyFunc1
...
MyFunc1()
```

How do we know which module MyFunc1 came from?

- **Or use an alias**

```
from MyModule import \
     (MyFunc1 as fred, MyFunc2 as jim)

fred()
jim()
```

Even worse.

Directories as packages

- **Keep related modules together in the same directory**
  - **Make sure the directory name is not the same as a Python system directory**
- **An `__init__.py` file is required**
  - **Might be empty**

Directory name / package name

```
import WorkingModules.MyModuleA

WorkingModules.MyModuleA.MyFunc1()
```

- **May be nested**
  - **Each nested sub-directory should have a `__init__.py` file**
  - **Each is just another name in the hierarchy**

Writing a module

- **No special header or footer required in the file**
  - **Just write your code without a 'main'**
  - **Default documentation is generated and available through help()**
- **Conventions with underscores - reminder**
  - **Names beginning with one underscore are private to a module**
    - Includes function names
  - **Names beginning and ending with two underscores have a special meaning**
- **Name of the module is available in __name__**

```
def MyFunc1():
    print "Hello from", __name__
```

Module documentation

- **Docstring for the module must be at the (very) start**
  - **Or explicitly assigned to __doc__**
  - **Used by the pydoc utility to generate documentation files**
  - **A default help format is provided**

```
>>> help(MyModuleA)
Help on module MyModuleA:

NAME
    MyModuleA

FILE
    c:\qa\python\mydemos\demomodules\mymodulea.py

DESCRIPTION
    This is a test module containing one
    function, MyFunc1

FUNCTIONS
    MyFunc1()
        MyFunc1 has no parameters and prints 'Hello'

DATA
    var1 = 42
```

Module docstring

Function docstring

Distributing python packages

- **Folder**
  - **distutils with sdist**
    - Creating a compressed file (zip/gz)
    - Install as a directory
- **Egg**
  - **setuptools with sdist**
    - Creating a compressed file
    - Install as egg
- **Wheel**
  - **setuptools with bdist_wheel**
    - Creating a whl file
    - Install with pip

Distributing libraries - distutils

- **There is a standard way of organising your files**
  - **Described in setup.py**

```
PyDealerPickCard/
    README.txt
    Documentation.txt
    libcard.py
    Showcard/
        __init__.py
        Showcard.py
    simple.py
    Bitmaps
    QA.ico
```

```
from distutils.core import setup
from glob import glob

setup(
    name = "PyDealerPickcard",
    version = "1.0",
    author = "QA",
    author_email = "QA.com",
    py_modules = ['libcard'],
    packages = ['Showcard'],
    scripts = ['simple.py'],
    data_files = [
        ('Bitmaps',glob('Bitmaps/*')),
        ('.', ['qa.ico'])],
    )
```

Distributing libraries - distutils

- Enables programs, modules, and packages to be bundled and unbundled in a standard way
  - Part of the standard library
- Based on setup.py written by the distributer
- Creating a distribution
  - Compressed file is placed into sub-directory `./dist`

```
$ python setup.py sdist
```

- Installing a distribution

```
$ tar xvf product-1.0.tar.gz
$ cd product-1.0
$ python setup.py install --record files.txt
```

Distributing Egg with setuptools

```
$ pip install setuptools
```

- **in setup.py**

```
from setuptools import setup
```

- **Creating a distribution with for *egg* format**
  - **Compressed file is placed into sub-directory `./dist`**

```
$ python setup.py sdist
```

- **Installing a distribution**

```
$ tar xvf product-1.0.tar.gz
$ cd product-1.0
$ python setup.py install --record files.txt
```

  - **Will install the *egg* file as bundle**

Distributing Wheel with setuptools

```
$ pip install wheel
$ pip install twine
```

- **Creating a distribution with *wheel* format**
  - **Bundle file is placed into sub-directory `dist`**

```
$ python setup.py bdist_wheel
```

- **Installing a distribution**

```
$ pip install demopack-1.3-py2-none-any.whl
```

  - **Will install the *whl* file (extracted)**

https://test.pypi.org/legacy/</code

Publishing Wheel to index

- **Main Python package index**
  - ***https://pypi.python.org/pypi***
  - **Testing site: *https://testpypi.python.org/pypi***
- **Create *.pypirc* file**
  - **Linux : ~/.pypirc**
  - **Windows: add to HOME env the path where .pypirc exists**
- **Publish**

```
[distutils]
index-servers=
pypi
Testpypi
[testpypi]
repository = https://test.pypi.org/legacy/
username = [uname]
password = [pass]
[pypi]
repository = https://pypi.python.org/pypi
username = [uname]
password = [pass]
```

```
$ python setup.py register -r
          https://testpypi.python.org/pypi

$ twine upload dist/* -r testpypi

$ pip install -i https://testpypi.python.org/pypi/ [MyPyPackName]
```

Wheel (2012) vs Egg (2004)

- Both are packaging formats
  - support install artifact that doesn't require building or compilation
- Wheel is currently considered the standard for built and binary packaging for Python
- Wheel is a distribution format, Egg was both a distribution format and a runtime installation format (if left zipped), and was designed to be importable.
- Wheel archives do not include .pyc files. compatible with Python 2 and 3
- Wheel is versioned.

Summary

- Writing a module in Python is simple
  - Just a bunch of code in a file
- Python loads modules based on sys.path
- Import a module using import
  - Can also specify importing names into our namespace
- Directories can be packages
  - Require the `__init__.py` file
- Python supports module documentation
  - docstrings
- There are several features and base modules to assist testing

Python debugger

- **Can be run from a script**

```
import sys
import pdb

sys.path.append('./DemoModules')
from person import Person

pdb.run ('me = Person ("Fred Bloggs", "m")')
print 'This is me',me
```

```
C:\QA>thing.py
> <string>(1)<module>()
(Pdb) s
--Call--
> c:\qa\demomodules\person.py(3)__init__()
-> def __init__ (self, name, gender):
(Pdb) s
```

- **Or from the command-line**

```
C:\QA>python -m pdb thing.py
```

Python profiler

- **The cProfile module**
  - **Profile a specific function from a script**

```
import mymodule
import cProfile
cProfile.run('mymodule.start()', 'start.prof')
```

Save statistics to this file (optional)

Default: display statistics to stdout

  - **Or the whole script from the command-line**

```
C:\QA>python -m cProfile thing.py
```

- **Analyse the output file using pstats shell**

```
C:\QA>python -m pstats start.prof
Welcome to the profile statistics browser.
% help
```