Python language: modules

The FOSSEE Group

Department of Aerospace Engineering
IIT Bombay

Mumbai, India

Outline

Modules

- Organize your code
- Collect similar functionality
- Functions, classes, constants, etc.

Modules

- Define variables, functions and classes in a file with a `.py` extension
- This file becomes a module!
- File name should be valid variable name

Modules . . .

- The **`import`** keyword “loads” a module
- One can also use:
  `from module import name1, name2`
  where **`name1`** etc. are names in the module, **`module`**
- **`from module import *`** — imports everything from module, use only in interactive mode

Note on module file names

- Should start with a letter or underscore
- Can use _ (underscore) and numbers
- No . allowed
- No spaces or special characters

Test

- `1_script.py`
- `script_1.py`
- `one11.py`
- `_one11.py`
- `one script.py`
- `one,script;xxx.py`
- `one.two.py`

Modules: example

```
# --- fib.py ---
some_var = 1
def fib(n):
    """Print Fibonacci series up to n.
    """
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
# EOF
```

Modules: example

```
>>> import fib
>>> fib.fib(10)
1 1 2 3 5 8
>>> fib.some_var
1
```

Python path

- In IPython type the following

```
import sys
sys.path
```

- List of locations where python searches for a module
- `import sys` – searches for file `sys.py` or dir `sys` in all these locations
- Modules can be in any one of the locations
- Current working directory is one of the locations
- Can also set `PYTHONPATH` env var

Running a module

- Let us say we want to run some code in **`fib.py`** so we get:

```
$ python fib.py
1 1 2 3 5 8
```

Modules: Running a module

```
# --- fib.py ---
some_var = 1
def fib(n):
    """Print Fibonacci series up to n.
    """
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b

fib(10)
# EOF
```

Modules: example

```
>>> import fib
1 1 2 3 5 8
```

- So the code is called even when we import the module
- We do not want this!

Using `__name__`

```
import fib
```

- The import is successful
- But **`fib(10)`**, gets run
- Add it to the following `if` block

```
if __name__ == "__main__":
    fib(10)
```

- Now the script runs properly
- As well as the import works; the code is not executed
- `__name__` is local to every module and is equal to `'__main__'` only when the file is run as a script.

Using `__name__`

```
import fib
```

- The import is successful
- But **`fib(10)`**, gets run
- Add it to the following `if` block

```
if __name__ == "__main__":
    fib(10)
```

- Now the script runs properly
- As well as the import works; the code is not executed
- `__name__` is local to every module and is equal to `'__main__'` only when the file is run as a script.

Modules: using __name__

```
# --- fib.py ---
def fib(n):
    """Print Fibonacci series up to n.
    """
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b

print(__name__)
if __name__ == '__main__':
    fib(10)
# EOF
```

Modules: using `__name__`

```
>>> import fib
```

Try this:

```
$ python fib.py
```

Stand-alone scripts

Consider a file `f.py`:

```
#!/usr/bin/env python
"""Module level documentation."""
# First line tells the shell that it should use Python
# to interpret the code in the file.
def f():
    print "f"

# Check if we are running standalone or as module.
# When imported, __name__ will not be '__main__'
if __name__ == '__main__':
    # This is not executed when f.py is imported.
    f()
```

Standard library

- Operating system interface: **`os`**, **`os.path`**
- System, Command line arguments: **`sys`**
- Regular expressions: **`re`**
- Math: **`math`**, **`random`**
- Internet access: **`urllib`**, **`smtplib`**
- Data compression: **`zlib`**, **`gzip`**, **`bz2`**, **`zipfile`**, and **`tarfile`**
- Unit testing: **`doctest`** and **`unittest`**
- And a whole lot more!
- Check out the Python Library reference: `http://docs.python.org/lib/lib.html`

Summary

- Creating modules
- Importing modules
- Running modules as scripts
- `__name__`