Modules in Python

What are Modules?

Modules are files containing Python definitions and statements (ex. *name*.py)

A module’s definitions can be imported into other modules by using “import *name*”

The module’s name is available as a global variable value ‘__name__’

To access a module’s functions, type *“name.function()”*

More on Modules

- Modules can contain executable statements along with function definitions
- Each module has its own private symbol table used as the global symbol table by all functions in the module
- Modules can import other modules
- Each module is imported once per interpreter session
  - *reload(name)*
- Can import names from a module into the importing module's symbol table
  - *from mod import m1, m2* (or **)*
  - *m1()*

Executing Modules

python name.py <arguments>

- Runs code as if it was imported
- Setting *_name_ == "_main_" the file can be used as a script and an importable module*

The Module Search Path

The interpreter searches for a file named *name.py*

- Current directory given by variable *sys.path*
- List of directories specified by **PYTHONPATH**
- Default path (in UNIX - *.:/usr/local/lib/python*)

Script being run should not have the same name as a standard module or an error will occur when the module is imported

“Compiled” Python Files

- If files *mod.pyc* and *mod.py* are in the same directory, there is a byte-compiled version of the module *mod*
- The modification time of the version of *mod.py* used to create *mod.pyc* is stored in *mod.pyc*
- Normally, the user does not need to do anything to create the *.pyc* file
- A compiled *.py* file is written to the *.pyc*
    - No error for failed attempt, *.pyc* is recognized as invalid
- Contents of the *.pyc* can be shared by different machines

Standard Modules

- Python comes with a library of standard modules described in the Python Library Reference
- Some are built into interpreter
- *>>> import sys*

  >>> sys.s1

  '>>> '

  >>> sys.s1 = 'c> '

  c> print 'Hello'

  Hello

  c>
- *sys.path* determines the interpreters's search path for modules, with the default path taken from **PYTHONPATH**
  - Can be modified with append() (ex. *Sys.path.append('SOMEPATH')*

The *dir()* Function

Used to find the names a module defines and returns a sorted list of strings

- *>>> import mod*

  >>> dir(mod)

  ['_name_', 'm1', 'm2']

Without arguments, it lists the names currently defined (variables, modules, functions, etc)

Does not list names of built-in functions and variables

- Use _*builtin*_to view all built-in functions and variables

Packages

- “dotted module names” (ex. *a.b)*
  - Submodule *b* in package *a*
- Saves authors of multi-module packages from worrying about each other’s module names
- Python searches through *sys.path* directories for the package subdirectory
- Users of the package can import individual modules from the package
- Ways to import submodules
  - *import sound.effects.echo*
  - *from sound.effects import echo*
- Submodules must be referenced by full name
- An *ImportError* exception is raised when the package cannot be found

Importing * From a Package

* does not import all submodules from a package

Ensures that the package has been imported, only importing the names of the submodules defined in the package

import sound.effects.echo

import sound.effects.surround

*from sound.effects import **

Sources

http://docs.python.org/tutorial/modules.html