What is a module?

- Technically, it is any file containing Python code
- Can define functions, classes, and variable
- Can contain runnable code
- Think of it as similar to a library
- Module name is filename without `.py` extension

A small example

```python
# spam.py
a = 37

def foo():
    print('foo: a = {}'.format(a))

def bar():
    print('bar: calling foo')
foo()

class Spam(object):
    def grok(self):
        print('Spam.grok')
```

Using our module

- To use the module `import` it in another program
- This causes the following to happen:
  - Executes code contained within the module
  - Creates a namespace that refers to the container for all objects defined in the module
  - Creates a name within the code that imported the module that refers to the module namespace
Using our module

```python
import spam

x = spam.a
print(x)  # prints 37
spam.foo()  # calls foo function
s = spam.Spam()
    # creates Spam instance
s.grok()  # invokes grok method
```

More about import

- As we know, we can change the name that refers to a module using the as qualifier
  - `import spam as sp`
- Using this we can be clever and make code more general
  ```python
  if format == 'xml':
      import xmlreader as reader
  elif format == 'csv':
      import csvreader as reader
  data = reader.read_data(filename)
  ```

Selective import

- We've also seen use of the from statement
  ```python
  from spam import foo
  foo()  # calls the function
  spam.foo()  # error: no spam namespace
  ```
- As we see in example above, using from does not create a namespace
  - Items are added to the current namespace
  - Does not change their scoping rules (see example on next slide)

Understanding Selective import

```python
from spam import foo

a = 42
foo()  # prints 37
```

Reference to `foo` is in current namespace but within `foo`, `a` is still bound to the global variable in the module in which foo was defined.
Understanding Selective import

from spam import bar

def foo():
    print(‘This is new foo’)
    bar()  # call to foo within bar
    # is to spam.foo

Logic is the same: call to foo in bar is still reference to spam.foo even though bar is in current namespace

Selective import with *

• Wildcard can be used with the from statement
  from spam import *
  foo()  # calls the function
  spam.foo()  # error: no spam namespace

• As we see in example above, using from does not create a namespace
  • Items are added to the current namespace

Selective import with *

• As the author of a module, you can control what is exported with *
  
  # spam.py module
  __all__ = [‘bar’, ‘Spam’]  # names exported with *

Running as Main Program

• Modules can make good use of __name__
  • Use the if case to run code that tests module functionality
    
    # check if running as program rather than module
    if __name__ == ‘__main__’:
        # running as main – put test or sample code here
    else:
        # no, imported as a module
Module Search Path

- The import path is contained in `sys.path`
- First entry is empty string for current directory
- We can modify the path by adding additional elements
  - directories – fully specified
  - zip files – can contain multiple modules

```python
import sys  # loads sys module
sys.path.append('/Users/mathias/my_python_libs')
sys.path.append('new_modules.zip')
```