UNIVERSITY of WISCONSIN
LA CROSSE
COMPUTER SCIENCE

CS 224 Introduction to Python

Modules

1

What is a module?

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

2

A small example

```
# 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')
```

3

Using our module

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

4

Using our module

```
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
```

5

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

```
if format == 'xml':
    import xmlreader as reader
elif format == 'csv':
    import csvreader as reader

data = reader.read_data(filename)
```

6

Selective `import`

- We've also seen use of the `from` statement

```
from spam import foo
foo()                          # calls the function
spam.foo()                     # error: no spam namespace
```

- As we see in the 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)

7

Understanding Selective `import`

```
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

8

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 as on previous slide: call to `foo` in `bar` is still reference to `spam.foo` even though `bar` is in current namespace

9

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, even though we are importing everything in `spam`, `from` does not create a namespace
  - Items are added to the current namespace

10

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 *
```

- Does not prevent direct importation of elements not in the list. The following is valid even though foo is not in the list:

```
from spam import foo
```

11

Selective `import` with *

- As the author of a module, you can control what is exported with *

```
# spam.py module
__all__ = ['bar', 'Spam', 'dada']
```

- Results in an error since there is no attribute `'dada'` in the module

12

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:
    # imported as a module – do anything needed
    # in this case
```

13

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

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

14