I. Namespaces & Scope

II. Advanced Libraries for Scientific Computing with Python

D. A. Sirianni

Institute for Data Engineering & Science
Georgia Institute of Technology

16 July, 2017

Overview

Primer

What will the following code, `scope_test.py`, print?

```python
some_name = 'Bob'

def print_name():
    some_name = 'Alice'
    print('In here, the name is %s' % some_name)

print('Out here, the name is %s' % some_name)
print_name()
```

Primer

What will the following code, `scope_test.py`, print?

```python
some_name = 'Bob'

def print_name():
    some_name = 'Alice'
    print('In here, the name is %s' % some_name)

print('Out here, the name is %s' % some_name)
print_name()
```

Answer:

```
$ python scope_test.py

Out here, the name is Bob.
In here, the name is Alice.
```

Some Definitions

Namespace

A *namespace* is a mapping of name to object, as in a Python dictionary:

```
space1 = {'name1': var1, 'name2': var2, ...}
space2 = {'name0': var1, 'name3': var2, 'name4': var4, ...}
```

Note: Namespaces are completely independent, and their names have no relation to one another. Furthermore, they can have different lifetimes.

Scope

Scope is the textual region of a Python program where a namespace is directly accessible.

Scope Regions: Local & Global

```python
# Here is the global (G) scope
global_var = "This is a global variable"

def my_function():
    # This is the local (L) scope
    local_var = "This is a local variable"
```

Scope Regions: Local & Global

```python
# Here is the global (G) scope
global_var = "This is a global variable"

def my_function():
    # This is the local (L) scope
    local_var = "This is a local variable"
```

Namespace Precedence

When Python searches for the object associated with a particular name, it proceeds from $L \rightarrow G$; this is referred to as their *namespace precedence*.

Scope Regions: Enclosing & Built-In

```python
# Here is the global (G) scope
global_var = "This is a global variable"

def outer_function():
    # This is the enclosing (E) space
    enclosing_var = "This is a nonlocal variable"

    def inner_function():
        # This is the local (L) space
        local_var = "This is a local variable"

# Some names are built-in (B) to Python
print(type(global_var))

def type():
    print("typetypetypetype")

type()
```

Namespace Precedence: The LEGB Rule

```python
def set():
    def do_local():
        spam = "local spam"
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"
    def do_global():
        global spam
        spam = "global spam"
    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

print("In global scope:", spam)
```

Namespace Precedence: The LEGB Rule

```
$ python legb.py

After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Namespace Precedence: The LEGB Rule

```
$ python legb.py

After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

The LEGB Rule

Object namespaces have the following order of precedence:

$$\text{Local} \rightarrow \text{Enclosed} \rightarrow \text{Global} \rightarrow \text{Built-In}$$

this is referred to as the *LEGB Rule*.

Namespaces of Modules and Classes

Our module, `module_test.py`:

```python
def distance(coords1, coords2):
    total = 0.0
    for k in list(len(coords1)):
        total += (coords1[k] - coords2[k]) ** 2
    return (total ** 0.5)
```

Namespaces of Modules and Classes

Our module, `module_test.py`:

```python
def distance(coords1, coords2):
    total = 0.0
    for k in list(len(coords1)):
        total += (coords1[k] - coords2[k]) ** 2
    return (total ** 0.5)
```

Friend's script, calling our module:

```python
from module_test import *

def distance(a, b):
    print('The distance between points A and B is %4.3f' % distance(a,b))

point1 = [0.0, 4.5, 1.4]
point2 = [3.3, 2.0, 0.7]
distance(point1, point2)
```

Namespaces of Modules and Classes (cont'd)

Problem: `from module import *` brings all code from `module.py` into the current namespace.

Solutions:

```
import module           # Imports module as separate namespace
import module as mod    # Gives module namespace a nickname
```

Friend's script, calling our module:

```
import module_test as dom

def distance(a, b):
    print('The distance between points A and B is %4.3f' % dom.distance(a,b))

point1 = [0.0, 4.5, 1.4]
point2 = [3.3, 2.0, 0.7]
distance(point1, point2)
```

I. Namespaces & Scope

II. Advanced Libraries for Scientific Computing with Python

D. A. Sirianni

Institute for Data Engineering & Science
Georgia Institute of Technology

16 July, 2017

Scientific Computing...

...is a rapidly growing, multidisciplinary field that uses advanced computing capabilities to understand and solve complex problems. It is an area of science which spans many disciplines, but at its core it involves the development of models and simulations to understand natural systems.[1]

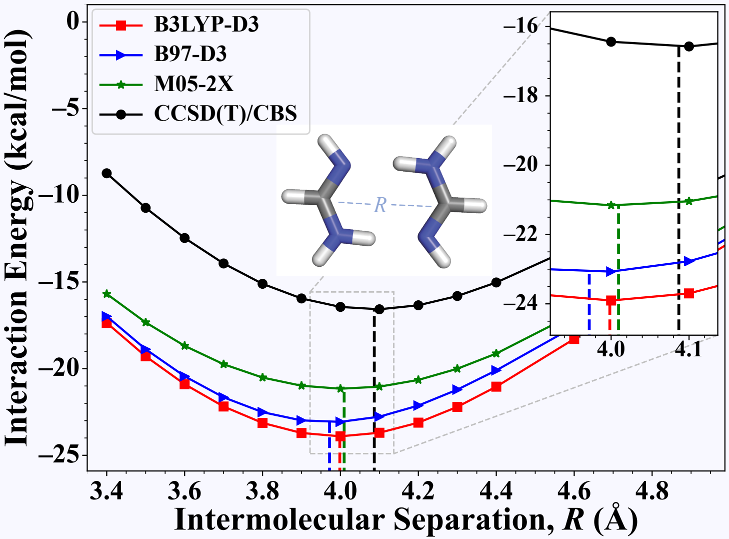
- Using industry-specific, fully-featured programs
- Integrating existing/available tools to solve specific problems
- Writing custom software routines to perform simulations, etc.

[1] From Wikipedia, article "Computational Science." https://en.wikipedia.org/wiki/Computational_science

Python Libraries for Scientific Computing

- SciPy: Diverse capabilities for scientific computing
- NumPy: Numerical linear algebra routines
- Pandas: Database capabilities
- Scikits: Domain-specific tools in a variety of fields
- Matplotlib: Generates publication-quality plots & figures
- Many others:
  `https://wiki.python.org/moin/NumericAndScientific`

An Illustrative Case Study: Curve Minima Interpolation


B3LYP-D3
B97-D3
M05-2X
CCSD(T)/CBS
R
Interaction Energy (kcal/mol)
Intermolecular Separation, R (Å)