

CS 2316 Data Manipulation for Engineers

Modules and Programs

Christopher Simpkins

`chris.simpkins@gatech.edu`

Python Programs

Python computations are organized as

- modules,
- packages, and
- programs.

We've already used some modules, now we'll learn what they are, how they're organized in packages, and how to write Python programs that can be run on their own, not just entered in the Python command shell.

Modules

A module is a file with Python code. Python includes many standard modules, such as `turtle`:

```
>>> import turtle
>>> turtle.left(90)
>>> turtle.forward(80)
>>> turtle.right(90)
>>> turtle.forward(80)
>>> turtle.left(143.14)
>>> turtle.forward(50)
>>> turtle.left(73.73)
>>> turtle.forward(50)
>>> turtle.left(98.14)
>>> turtle.forward(113.14)
>>> turtle.left(135)
>>> turtle.forward(80)
>>> turtle.left(135)
>>> turtle.forward(113.14)
>>> turtle.left(135)
>>> turtle.forward(80)
```

You can see what this draws by running [house.py](#).

Importing Modules

importing a module means getting names from the module into scope. When you import a module, you can access the modules components with the dot operator as in the previous example.

```
>>> import math
>>> math.sqrt(64)
8.0
```

You can also import a module and give it an alias: `import <module> as <local-name>`

```
>>> import math as m
>>> m.sqrt(64)
8.0
```

Importing into Local Scope

Remember that importing brings names into the scope of the import. Here we import the `math` module into one function.

```
>>> def hypotenuse(a, b):  
...     import math  
...     return math.sqrt(a*a + b*b)  
...  
>>> hypotenuse(3, 4)  
5.0  
>>> math.sqrt(64)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'math' is not defined
```

Importing Names from Modules

You can import only certain names from a module:

```
>>> from math import sqrt
>>> sqrt(64)
8.0
>>> floor(1.2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'floor' is not defined
```

Or all names from a module:

```
>>> from math import *
>>> floor(1.2)
1
>>> sin(0)
0.0
>>> sin(.5 * pi)
1.0
```

Notice that with this syntax you don't have to use a fully-qualified name, e.g., `module.name`

Module Search Path

Just as an operating system command shell searches for executable programs by searching the path, Python finds modules by searching directories. The module search path is stored in `sys.path`:

```
>>> import sys
>>> sys.path
['',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4.zip',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/platform',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/lib-dynload',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages']
```

Notice that the current directory, represented by the `"` at the beginning of the search path.

Pretty Printing

You can get "prettier" printing with the `pprint` module:

```
>>> import pprint as pp
>>> pp.pprint(sys.path)
['',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python34.zip',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4',
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/plat-
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/lib-c
 '/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-
```


Packages

Modules are just Python files. The directory path from some `sys.path` root to a Python source file is called a package. Make a directory called `hanglib` and download this file into it: [drawing.py](#)

```
>>> import hanglib.drawing
>>> hanglib.drawing.stand()
```

`hanglib` is a package¹ and `drawing` is a module within the `hanglib` package. Of course, you could import it as an alias to make coding more convenient:

```
>>> import hanglib.drawing
>>> hanglib.drawing.stand()
>>> import hanglib.drawing as draw
>>> draw.head()
```

Notice that, as in this example, we can make our own modules.

¹Technically you should have a `__init__.py` file in package directories, but that's not important for our basic use cases.

Programs

Take a look at the [drawing.py](#) file. Notice the `if` statement at the bottom:

```
# is this the main (top-level) module?
if __name__ == '__main__':
    stand()
    head()
    body()
    leftarm()
    rightarm()
    leftleg()
    rightleg()
    # Pause so the user can see the drawing before exiting.
    input('Press any key to exit.')
```

This makes the module a runnable Python program. It's similar to the `main` function or method from some other programming languages. With it we can import the file as a module to use its functions (or objects or variables), or run it from the command line.

Shebang!

Another way to run a Python program (on Unix) is to tell the host operating system how to run it. We do that with a "shebang" line at the beginning of a Python program:

```
#!/usr/bin/env python3
```

This line says "run python3 and pass this file as an argument." So if you have a program called `foo` with shebang line as above and which has been set executable (`chmod +x foo.py`), these are equivalent:

```
$ python3 foo.py
$ ./foo.py
```

Interactive Programs

The `input()` function Python reads all the characters typed into the console until the user presses ENTER and returns them as a string:

```
>>> x = input()
abcdefg1234567
>>> x
'abcdefg1234567'
```

We can also supply a prompt for the user:

```
>>> input('Give me a number: ')
Give me a number: 3
'3'
```

And remember, `input()` returns a string that we may need to covert to use.

```
>>> 2 * int(input("Give me a number and I'll double it: "))
Give me a number and I'll double it: 3
6
```

Command-line Arguments

When you run a Python program, Python collects the arguments to the program in a variable called `sys.argv`. Given a Python program (`arguments.py`):

```
#!/usr/bin/env python3

import sys

if len(sys.argv) < 2:
    print("You've given me nothing to work with.")
else:
    print(sys.argv[1] + "? Well I disagree!")
```

```
$ ./arguments.py Pickles
Pickles? Well I disagree!
[chris@lawn-128-61-62-82 ~/work/vcs/github/data-python/code]
$ ./arguments.py
You've given me nothing to work with.
```

Closing Thoughts

- Use modules and packages to organize large programs
- Import to minimize name conflicts
- Program from the "bottom up". Get module components to work in isolation, then put the modules together to form a program.