Modules and Programs

Python Programs

Python code organized in

- modules,
- packages, and
- scripts.

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

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.

```
1 >>> import math
2 >>> math.sqrt(64)
3 8.0
```

You can also import a module and give it an alias: import as

```
1 >>> import math as m
2 >>> m.sqrt(64)
3 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
```

But it's not available at the top level.

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

Importing Names from a Module

You can choose to 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 directories listed in the `PATH` environment variable, Python finds modules by searching directories. The module search path is stored in `sys.path`:

```
>>> import sys
>>> from pprint import pprint
>>> pprint(sys.path)
['',
 '/usr/lib/python38.zip',
 '/usr/lib/python3.8',
 '/usr/lib/python3.8/lib-dynload',
 '/home/chris/.local/lib/python3.8/site-packages',
 '/usr/local/lib/python3.8/dist-packages',
 '/usr/lib/python3/dist-packages']
```

Notice that the current directory, represented by the `''` at the beginning of the search path, is part of `sys.path`, which is why you can import modules located in your current directory.

Also, note use of `pprint`.

Writing Python Modules

A Python module is a file containing definitions. These definitions can include classes, functions or vairables.

Exercise 1

Python Scripts

A Python script is a file containing executable Python code. Our `hello.py` script from Day 1 is an example of a Python script. Note that a module can be a Python script if it contains code that executes whenever the module

`if __name__ == '__main__'`

Take a look at the draw.py file. Notice the if statement at the bottom:

```
1  # Is this the main (top-level) module?
2  if __name__ == '__main__':
3      stand()
4      head()
5      body()
6      leftarm()
7      rightarm()
8      leftleg()
9      rightleg()
10     # Pause so the user can see the drawing before exiting.
11     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:

```
1 #!/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`), thse are equivalent:

```
1 $ python3 foo.py
2 $ ./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 may need to be converted.

```
>>> 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

```
$ python args.py one 2 two + one
```

The `python` invocation above contains 6 command line arguments.

Command-line Arguments in Python

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):

```
1  #!/usr/bin/env python3
2  import sys
3
4  if len(sys.argv) < 2:
5      print("You've given me nothing to work with.")
6  else:
7      print(sys.argv[1] +"? Well I disagree!")
```

```
1  $ ./arguments.py Pickles
2  Pickles? Well I disagree!
3  $ ./arguments.py
4  You've given me nothing to work with.
```

Conclusion

Python Arguments