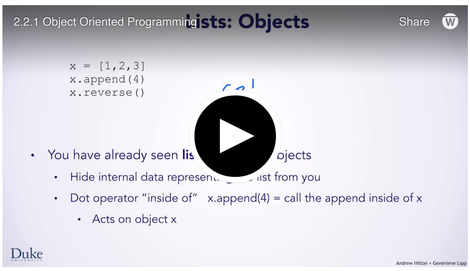
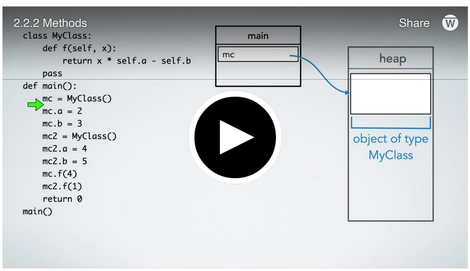
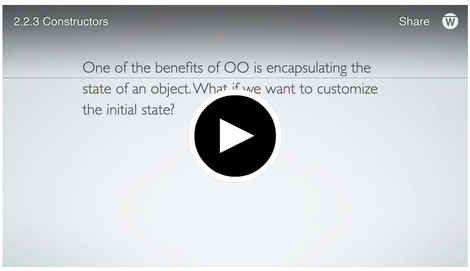
Object-Oriented Programming in Python



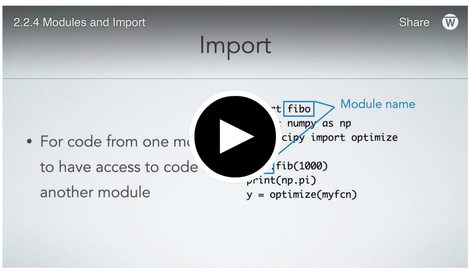
Methods



Constructors



Modules and Import



Python: Import Doesn't Reload Modules

As you learned in the previous video, "import" lets you load code from other modules—whether they are ones you wrote, or part of libraries that come with python. One annoying oddity of import in Python is that it does NOT reload a module if that module has changed. This "feature" is only a problem if you keep Python running while you (1) import a module, (2) change that module, and (3) try to "re-import" the module (which does nothing).

As an example, suppose you have two modules **point** and **circle** (as you will in upcoming assignments). Here, the Circle class (defined in module circle) makes use of the Point class (defined in module point) to represent its center, so point imports circle.

Let us suppose you write point, then go write circle. You evaluate the code (C-c C-c) and find that there is a bug. You trace this code back to something in point, and fix it. You go back to circle and hit C-c C-c again... but the bug is still there! This problem is quite frustrating and often confusing to novices. Even though you changed the code, point was not reloaded by re-evaluating circle (since Python considers point to be already loaded).

So how can you deal with this?

One option (which is, in our opinion the simplest) is to just quit Python by either typing **quit()** or control-d at the Python prompt. Then hit C-c C-c in circle. Now, Python emacs will start a fresh Python interpreter, and it won't have any modules loaded, so the import point will read the new point. This approach works for all modules (without you having to think about what changed and what did not).

If you want to reload a module without restarting Python, how you do it depends on the version of Python you are using. For Python versions above 3.4, you do

```
import importlib
importlib.reload(point)
```

Note that there are no quotes around point. It is not the name of the module, but the module itself (which was bound to the name point when imported).

For Python 3 before 3.4, you would do

```
import imp
imp.reload(point)
```

And for Python 2.x you would do

```
reload(point)
```

We don't expect you to remember all the different ways to reload a module (and in fact, we prefer to just hit control d at the Python prompt, then C-c C-c in our code to start a fresh Python when stuff like this happens). We do want you to know about this potential pitfall so you do not get frustrated and stuck when you have changed something but the module is not reloaded, so your changes are not showing up.

Day 7, Part 1

Now it is time to do **14_point** and **15_circle** on the Mastery Learning Platform. Login to the course server, go to **14_point** in your git repository, and read the README for directions. When you have passed that assignment, complete **15_circle** on the MLP as well. When you've completed both, return to Sakai and continue with the content here.