Chapter 1: Serialization

Summary:
Have you ever cooked before? I’m betting you have. I’m also betting you’ve had leftovers that you might have frozen or otherwise stored for later. Well, in this chapter, we’re going to discuss how to do the same thing with objects in Python!

As you create projects with more and more complex classes and objects, you’ll eventually find yourself wanting to save the state of the instances you create. In other words, you want to be able to keep the values of their variables and "state" and recreate them with no hassle. That’s where serialization comes in to save the day. I like to think of serialization as freezing food for later—or as you’ll soon see in Python, we’ll call it pickling. When you’re ready to save the food, or object, for later use, you just serialize it and store it away; then later when you want to use it again, you thaw it out--or in our case deserialize it!

Example 1-1 (simple_serialize.py):
```
import pickle
awesome_data = list(range(10))
pickle.dump(awesome_data, open("awesome_save.sav", "wb"))
```
-
First thing's first, we have to import our package containing needed sorcery, pickle. This package comes installed with Python and is awesome for easy serialization/deserialization. Then we generate some sample data that is simply a list of the numbers 0-9. Finally, we want to save

this object, so we just "dump" it in a file called 'awesome_save.sav.' You typically want to use the "wb" mode on the open function, which means "write binary" so it pickle will write the data as binary. Note: you can call this file whatever you want and give it any extension or none at all; I simply like .sav for my "save" files.

Now you ask, "Okay...but how do I get my awesome data back?" Here comes the deserialization part:

Example 1-2 (simply_deserialize.py):

```
import pickle
awesome_data = pickle.load(open("awesome_save.sav", "rb"))
print(awesome_data)
```

-

So the only difference is, now we're "loading" instead of "dumping." This time, the open function will use the "read binary" mode, since we wrote the save file as binary. Thanks to the print() statement at the end, we can verify that our awesome data is indeed intact.

"That's just a list though. What about my super complex class I built? How do I save all of its awesome data?"

Have no fear, Python is here:

Example 1-3 (complex_serialize.py):