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!

Okay, let's jump right into it then, shall we!

Example 1-1 (simple_serialize.py):
```
import pickle
simple_data = list(range(10))
pickle.dump(simple_data, open("simple.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 simple data that is just a list of the numbers 0-9. Finally, we want to save this object, so we just "dump" it in a file called 'simple.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 data back?" Here comes the deserialization part:

Example 1-2 (simple_deserialize.py):

```
import pickle
simple_data = pickle.load(open("simple.sav", "rb"))
print(simple_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 simple 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. First, we'll need a "complex class" like the following:

Example 1-3 (complex.py):

```
class ComplexThing:
    def __init__(self, name, data):
        self.name = name
        self.data = data

    def __str__(self):
        return self.name + ": " + str(self.data)
---

And then the serialization bit:

---
Example 1-4 (complex_serialization.py):
import pickle
from complex import ComplexThing

# Create the instance
awesome_thing = ComplexThing("awesomeness", 15)
print("Before Pickling:")
print(awesome_thing)

# Pickle the instance
pickle.dump(awesome_thing, open("complex.sav", "wb"))

# Load the instance from save
loaded = pickle.load(open("complex.sav", "rb"))

# Print the loaded instance to verify intact data
print("\nAfter Loading from Pickle: ")
print(loaded)
---

You mean to pickle an instance of my custom class, I simply
do the same thing as before? Yep, it's that easy. This
```

example might seem different, but the serialization is the same; you just pass in the instance of your class to pickle.dump() instead of the simple data from Example 1-1. Please note, however, that with very large or nested data, you might need to rethink your storage method (I highly recommend some research on the Python "shelve" package for this).

Wrap Up:
So now you know how to "freeze" your objects for later, and then bring them back to life when the time calls! This topic may seem boring at first, but trust me when I say you'll thank me later. An easy application of serialization is a configuration class for say...a text editor you created (totally not going to make one of those later in this book...[hint of sarcasm]). Using serialization, you can save the values of all the settings the user has set for the application, and when they hit "Save Settings" it dumps the pickled data into a save file. Then, whenever the user starts the application up, it checks for a save file and loads the settings from it if it exists.
