Diving In

WE'LL COVER THE FOLLOWING

- A Quick Note About The Examples in This Chapter

On the surface, the concept of serialization is simple. You have a data structure in memory that you want to save, reuse, or send to someone else. How would you do that? Well, that depends on how you want to save it, how you want to reuse it, and to whom you want to send it. Many games allow you to save your progress when you quit the game and pick up where you left off when you relaunch the game. (Actually, many non-gaming applications do this as well.) In this case, a data structure that captures "your progress so far" needs to be stored on disk when you quit, then loaded from disk when you relaunch. The data is only meant to be used by the same program that created it, never sent over a network, and never read by anything other than the program that created it. Therefore, the interoperability issues are limited to ensuring that later versions of the program can read data written by earlier versions.

For cases like this, the `pickle` module is ideal. It's part of the Python standard library, so it's always available. It's fast; the bulk of it is written in C, like the Python interpreter itself. It can store arbitrarily complex Python data structures.

What can the `pickle` module store?

- All the native datatypes that Python supports: booleans, integers, floating point numbers, complex numbers, strings, `bytes` objects, byte arrays, and None.
- Lists, tuples, dictionaries, and sets containing any combination of native datatypes.

- Lists, tuples, dictionaries, and sets containing any combination of lists, tuples, dictionaries, and sets containing any combination of native datatypes (and so on, to the maximum nesting level that Python supports).
- Functions, classes, and instances of classes (with caveats).

If this isn't enough for you, the `pickle` module is also extensible. If you're interested in extensibility, check out the links in the Further Reading section at the end of the chapter.

A Quick Note About The Examples in This Chapter

This chapter tells a tale with two Python Shells. All of the examples in this chapter are part of a single story arc. You will be asked to switch back and forth between the two Python Shells as I demonstrate the `pickle` and `json` modules.

To help keep things straight, open the Python Shell and define the following variable:

```
shell = 1
```

Keep that window open. Now open another Python Shell and define the following variable:

```
shell = 2
```

Throughout this chapter, I will use the `shell` variable to indicate which Python Shell is being used in each example.