Object Serialization

Serializing and Deserializing (Pickling andUnpickling)

We can store an object into a file, and retrieving it later from storage. Storing an object is called *serializing* it, and retrieving it is called *deserializing* it. Python uses a function called `pickle` to do this. So sometimes instead of saying we are serializing an object, we can say we are *pickling an object*; and instead of deserializing an object, we can say we are *unpickling an object*.

The `Dump` Method

To save an object to a byte file, `pickle` provides a `dump` method:

```
pickle.dump(object, file)
```

For example:

```
open("pickled_list.p", "wb") as MyFile:
pickle.dump(MyObject, MyFile)
```

So we open a byte file ("pickled_list.p") for writing, as MyFile, and serialize that object into MyFile.

The `Load` Method

To take a byte file and load it into an object, we have the `load` method:

```
object = pickle.load(file)
```

For example:

```
open("pickled_list.p", "rb") as MyFile:
MyNewObject = pickle.load(MyFile)
```

So open the byte file ("pickled_list.p") for reading, as MyFile, and load it into the object.

Example Program

```
import pickle
MyObject = ["a list", "containing", 5, "values including
another list", ["inner", "list"]]

with open("pickled_list.p", "wb") as MyFile:
    pickle.dump(MyObject, MyFile)
with open("pickled_list.p", "rb") as MyFile:
    MyNewObject = pickle.load(MyFile)
```

Using the `with` statement means that the file is automatically closed when finished.

The `Dumps` and `Loads` Methods

The `dumps` and `loads` behave much like their file-like counterparts, except they return or accept byte strings instead of file-like objects. The `dumps` method requires only one argument, the object to be stored, and it returns a serialized byte string object. The `loads` method requires a byte string object and returns the restored object.

```
import pickle
MyString = ["a list", "containing", 5, "values"]
DumpedString = pickle.dumps(MyString)
LoadedString = pickle.loads(DumpedString)
```

The variable `LoadedString` is uploaded as a byte string.

Object Serialization

Serializing Web Objects

To transmit object data over the web you need to use a recognised standard so that the sending and receiving classes will know what is being transmitted. There are many common standards, but the most common one is JSON ("jason"). JSON stands for JavaScript Object Notation, and is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. JSON is a language-independent data format, and the JSON filename extension is .json:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
       "streetAddress": "21 2nd Street",
       "city": "New York"
   }
}
```

The `Import` Statement

To deal with JSON web objects, Python provides a json library, so we say:

```
import json
```

The `Dump` and `Load` Methods

The `Dump` and `Load` methods work almost exactly the same way as their pickle counterparts, except that we are creating text files with valid JSON notation rather than byte files. So to save an object to a JSON file, `json` provides a `dump` method:

```
json.dump(object, file)
```

For example:

```
  open("MyFile.json", "w") as WriteFile:
  json.dump(MyObject, WriteFile)
```

So we open a text file ("MyFile.json") for writing, as WriteFile, and serialize that object into MyFile. And to take a text file and load it into a JSON object, we `load`:

```
object = json.load(file)
```

For example:

```
open("MyFile.json ", "r") as WriteFile:
MyNewObject = json.load(WriteFile)
```

So open the text file ("MyFile.json") for reading, as MyFile, and load it into the object.

The `Dumps` and `Loads` Methods

The `dumps` and `loads` behave much like their `pickle` counterparts:

```
import json
MyString = ["a list", "containing", 5, "values"]
DumpedString = json.dumps(MyString)
LoadedString = json.loads(DumpedString)
```

The variable `LoadedString` is uploaded as a byte string.