Using Python Dictionary as a database

by Scott Davidson

This guide discusses using Python's Dictionary object to access nested data.

Overview

There are many modern data structures that use a structured key:value pairs to describe objects and the data that is stored within them. A few popular ones are XML, JSON and Amazon S3(Dynamo).

The Dictionary object is used to hold a set of data values in the form of (key, value) pairs. The values can be any standard datatype including lists. This article may serve help understand how Python can be used to create and access nested information.

Creating a Key:Value datastore

Using Dictionaries, list and a key:values can be used together to create this datastore. Here is an example of a nested dictionary that stores many different items. In this case, we have a series of polylines representing various rooms for a medical office. Look closely at the bracket and parens that are used. The curly braces `{}` denote the a dictionary. The square brackets `[]` represent a list as a value in the `medical` key. The list in ‘medical’ actually contains a series of dictionaries for each individual office.

```
datastore = { "office": {
    "medical": [
      { "room-number": 100,
        "use": "reception",
        "sq-ft": 50,
        "price": 75
      },
      { "room-number": 101,
        "use": "waiting",
        "sq-ft": 250,
        "price": 75
      },
      { "room-number": 102,
        "use": "examination",
        "sq-ft": 125,
        "price": 150
      },
      { "room-number": 103,
        "use": "examination",
        "sq-ft": 125,
        "price": 150
      },
      { "room-number": 104,
        "use": "office",
        "sq-ft": 150,
        "price": 100
      }
    ],
    "parking": {
      "location": "premium",
      "style": "covered",
      "price": 750
    }
  }
}
```

Accessing the Datastore

There are many ways to access the data in this datastore:

```
print datastore["office"]["parking"]
```

This returns the `parking` dictionary object `{ "location": "premium", "style": "covered", "price": 750 }`

Knowing that the `value` for `medical` is a list. Use and index number to access any single book:

```
 print datastore["office"]["medical"][1]
```

This returns the dictionary object for room 100, reception.

The objects and values in the datastore can also be accessed with the `.get` method. The direct method shown above will return an error if a key does not exist. The `.get` method is a little safer. It will return a value or `None`. This is much safer if you are not sure the key is always present. The `isbn` key is a good example of this.

```
print datastore["office"]["law"] # this produces an error.
print datastore["office"].get("law")  #This will produce the value of None.
```

A convenient way to efficiently address a portion of the datastore is to assign the portion to a variable. In this case we can assign the list of books to a `spaces` variable:

```
spaces = datastore['office']['medical']
```

The variable is a reference to the object. Any changes made with `spaces` will also be reflected in the original datastore. Also, because `spaces` contains only the list of spaces in the datastore, it is quite easy to step through the spaces with a `for` statement. In the example below, the for loop is looking for a specific space then updates the price:

```
# Here is a method to find and change a value in the database.
for item in spaces:
    if item.get('use') == "examination" :
       item['price'] = 175

for item in datastore['office']['medical']: # This loop shows the change is
not only in books, but is also in database
    if item.get('use') == "examination" :
        print 'The {} rooms now cost {}'.format(item.get("use"),
item.get("price"))
```

Author: Scott Davidson