Assessment - Object-oriented programming

In this exercise, we'll create a few classes to simulate a server that's taking connections from the outside and then a load balancer that ensures that there are enough servers to serve those connections.

To represent the servers that are taking care of the connections, we'll use a Server class. Each connection is represented by an id, that could, for example, be the IP address of the computer connecting to the server. For our simulation, each connection creates a random amount of load in the server, between 1 and 10.

Run the following code that defines this Server class.

In [1]:

```
#Begin Portion 1#
import random

class Server:
    def __init__(self):
        """Creates a new server instance, with no active connections."""
        self.connections = {}

    def add_connection(self, connection_id):
        """Adds a new connection to this server."""
        connection_load = random.random()*10+1
        self.connections[connection_id] = connection_load
        # Add the connection to the dictionary with the calculated load

    def close_connection(self, connection_id):
        """Closes a connection on this server."""
        # Remove the connection from the dictionary
        del self.connections[connection_id]

    def load(self):
        """Calculates the current load for all connections."""
        total = 0
        # Add up the load for each of the connections
        for load in self.connections.values():
            total += load
        return total

    def __str__(self):
        """Returns a string with the current load of the server"""
        return "{:.2f}%".format(self.load())

#End Portion 1#
```

Now run the following cell to create a Server instance and add a connection to it, then check the load:

In [2]:

```
server = Server()
server.add_connection("192.168.1.1")

print(server.load())
```

```
8.153514930146258
```

After running the above code cell, if you get a **NameError** message, be sure to run the Server class definition code block first.

The output should be 0. This is because some things are missing from the Server class. So, you'll need to go back and fill in the blanks to make it behave properly.

Go back to the Server class definition and fill in the missing parts for the `add_connection` and `load` methods to make the cell above print a number different than zero. As the load is calculated randomly, this number should be different each time the code is executed.

Hint: Recall that you can iterate through the values of your connections dictionary just as you would any sequence.

Great! If your output is a random number between 1 and 10, you have successfully coded the `add_connection` and `load` methods of the Server class. Well done!

What about closing a connection? Right now the `close_connection` method doesn't do anything. Go back to the Server class definition and fill in the missing code for the `close_connection` method to make the following code work correctly:

In [3]:

```
server.close_connection("192.168.1.1")
print(server.load())
```

```
0
```

You have successfully coded the `close_connection` method if the cell above prints 0.

Hint: Remember that `del` dictionary[key] removes the item with key *key* from the dictionary.

Alright, we now have a basic implementation of the server class. Let's look at the basic LoadBalancing class. This class will start with only one server available. When a connection gets added, it will randomly select a server to serve that connection, and then pass on the connection to the server. The LoadBalancing class also needs to keep track of the ongoing connections to be able to close them. This is the basic structure: