Meridian Witt

1. In chapter 4: What happens in the client and in the server when the following line of code is executed: `PlayersList = new Mongo.Collection('players');`

Server side: A new collection is created inside the MongoDB database
Client side: Executes the code and created a local copy of the collection (allows user to interact with local copy)

2. In chapter 5: Show an example of the old-way of writing a template helper function. What warning message did you see in the browser console about using this old-way template?

Old way: Template.leaderboard.player format (look for a template with the name leaderboard and create a function called player)

The error shown in the textbook (I received no error but the text simply did not appear) says that this old way has been deprecated.

3. How does the new syntax for defining helper functions look like?
   What it allows us to do?
   Why is this approach better?

It allows us to define multiple helper functions in a single block of code. It is better because it keeps all the helper functions in one place and it is useful when there are many helper functions/templates

4. Move the `{{#each}}` block outside the template.
   What kind of error do you see in the browser console?
   What kind of error do you see in the server console?
   Which message is more useful?

I'm not sure where I should have moved the block, but I think that the error should occur in the browser console where the code is executed and a local copy is made. I did not see the error you are asking about.

5.

6. At the start of Chapter 6 (Events): If we click on the h1 element with the text Leaderboard, we don't get to see a message in the console, as it happens when we click on other parts of the page. Why?

The click statement only applies to things within the bounds of the leaderboard template.

7. At the end of Chapter 6: Try to add the dblclick event in your code, so that whenever you double-click on a list item, you can see on the console a message like in this screenschot. Remember that event handler functions can take a parameter, "event", through which, you'll be able to access the text of the clicked li item.

8. Include a screenshot of your app at the end of Chapter 7 (Sessions). Challenge question:In this chapter, the code for the "click" event was defined at the beginning and no code for highlighting a list item is in it. Why then do the list items change color whenever we click.

Called the selectedClass helper in the HTML li tag which created the color changing functionality

9. In Chapter 8, section "Advanced Operators Part 1", the update for incrementing by 5 points, replaced some elements entirely, for example see this screenshot. Show here the code you wrote to make this problem disappear.

```
'click .increment': function(){
        var selectedPlayer = Session.get('selectedPlayer');
        PlayersList.update(selectedPlayer, {$inc: {score:5}});
      }
```

10. Related to previous question: We want to replace the broken list elements (": 5"), to show the same text as before, for example, "Sophia : 5". Show here the code you can write in the console to fix this problem

I'm not sure how to write this in the console but it would used the selectedPlayer logic to get the correct object, use .update to then create a field for a name and $set to set the name to the appropriate name value.

11. At the end of Chapter 8, my app looks like in this screenshot. Describe in words what I have done to show the name in red. Then, try to implement this feature and show here the code for its implementation.

You wrapped the reference to the selectedPlayer name in a span tag with an ID that you edit in the CSS.

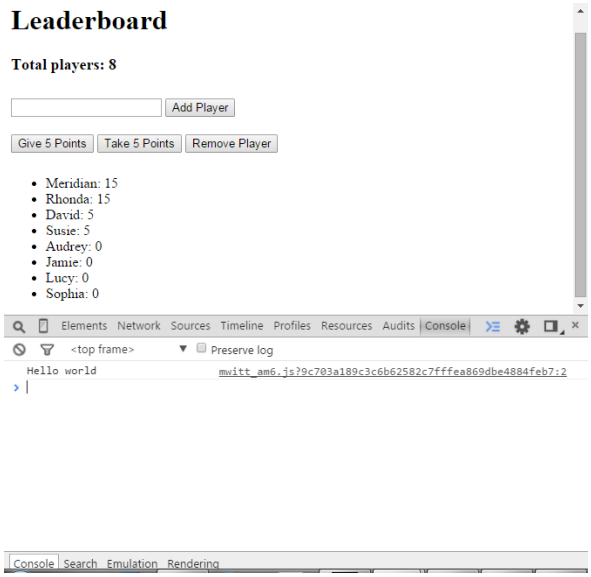
In CSS:

```
#selectedPlayer{
    color: red;
    font-weight: bold;
}
```

In HTML:

```
 <p>Selected Player: <span id="selectedPlayer">{{showSelectedPlayer.name}}</span></p>
```

12. At the end of Chapter 9 (forms), this is the state of my app, after having removed or added some players with the form. Make some modifications to your leaderboard this way and include the screenshot of your app here.

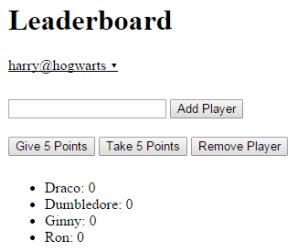


13. Once you have added accounts (end of Chapter 10), log in with a fictional name and add some new players. Here is a Hogwarts example. Show the screenshot of your app here.

14. What security problem has our current version of the app?
    How was that exposed?
    Which easy fix removes this problem?

Any person, regardless of who is signed in, can access the PlayersList through the console. It is exposed by default by Meteor with the autopublish package. Removing this package fixes the problem

15 After fixing the problems above using Meteor.Publish() and Meteor.Subscribe(), create two different user logins and add a few different players to each account. Then, run`PlayersList.find().fetch()` on the browser console and get a screenshot of your app at that point. This is my app's screenshot. Include your screenshot here.



16. In Chapter 12, what problem with our app will Meteor methods help solve?

To make the app more secure we removed the default insert and delete functionalities in the browser and created our own in the backend.

17.After you perform "meteor remove insecure", try to add a new player to the leaderboard. What message do you get in the console?
Then, try to increment points for a player. What message do you get in the console?

The message said it was restricted for both because that functionality was removed with the package.

18. Show here some examples of how `Meteor.call` was used during the Chapter 12. Can you explain the signature of this versatile method? How does it work?

```
Meteor.call('insertPlayerData', playerNameVar)
Meteor.call('removePlayerData', selectedPlayer);
```

It calls the method passed in the first argument on the object/variable in the second argument.

You can skip Chapter 13 in the book and move directly to Chapter 14 Deployment. However, do not deploy the app before adding your CSS styling to it, so that you can received the points for that requirement.