Introduction

For the form to work as expected, we need to get the value of the text field from the submit form event and into the `createPlayer` method. This will allow users to create players with unique name fields rather than every player being named David. To do this, pass the `playerName` variable into the `Meteor.call` statement as a second argument, then allow the `createPlayer` method to accept this argument:

```
Meteor.call('createPlayer', playerName);
```

```
createPlayer: function(playerName) {
}
```

Based on this change, we can now reference `playerName` from inside the method to reference the value that the user enters into the text field. This means we can pass the `playerName` variable into the name field. As a result, the Add Player form will work as expected, but users still won't be able to use the `insert` function from inside the console.

Something that might not be obvious is the fact that users can call methods from inside the console. For example, any user can execute the `createPlayer` method. And because the method accepts an argument, they can attach any value to the method. How would a user know that a `createPlayer` method even exists? Well, even after the application is deployed, users can simply view the source of the page and look through the JavaScript code.

Depending on how we structure the project, we can make it so some of the code isn't viewable in this way, but users will generally be able to view the code for the methods. As such, there's two problems we need to account for.

Using a Check Function

First, we need to make sure users can only pass a string into the method. This will prevent users from entering a numerical value, or a boolean value, or any other type of data that might stop the application from working as expected.

To do this, we can use a `check` function at the top of the method that allows us to check whether or not a certain value meets a set of requirements that we define.

To use the `check` function though, we need to first install the "check" package which can be done using the following command:

```
meteor add check
```

Noe inside this function, we need to pass through two arguments: A value and the object type that we're expecting this value to be. In this case, we'll pass through the `playerName` variable and we'll match it against the string object type:

```
check(playerName, String);
```

Because of this code, if a user tries to execute the method by passing through a value that is not a string, an error will be returned and the rest of the method won't be executed. The method will only execute if player name bar is a string. We'll talk more about check functions in another course, but for the time being, it's only important to understand because of this function, users won't be able to pass unwanted data into the method.

Fixing Issues

The second problem we need to account for is that currently, logged out users are able to call this method from the console. This means any user could fill the database with meaningless data that isn't attached to any particular account. To fix this, create a conditional around the `insert` function that checks the value of the `currentUserId` variable. It will return `true` if the current user is logged in:

```
if(currentUserId) {
    // ...
} else {
    console.log('Unauthorized');
}
```

As a result, the `insert` function will only execute if the current user is logged in, which prevents anonymous users from calling the method and adding meaningless data to the database.