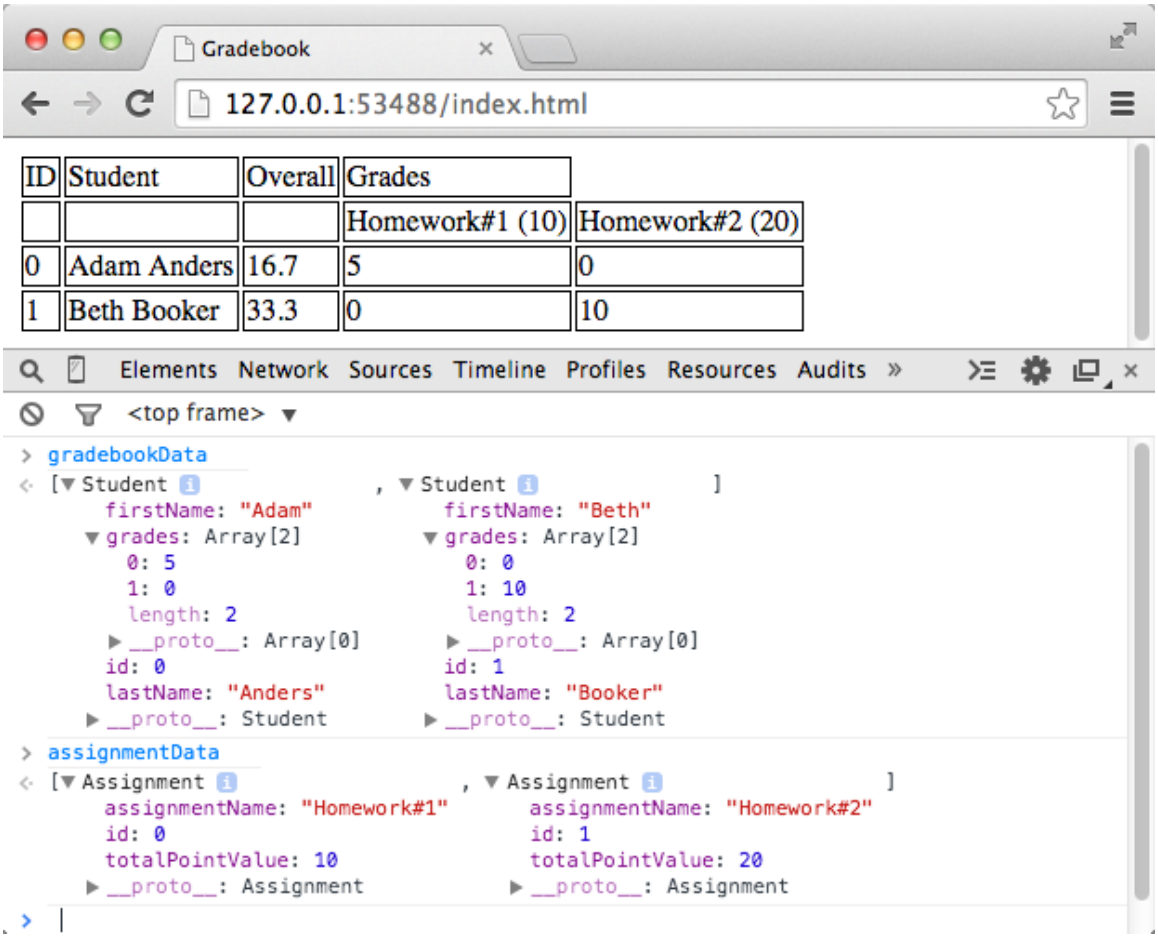
Workshop Week 2

In this week's workshop, we are going to review Arrays, Objects, and Functions by creating a simple grade recording application. Because we have not yet covered Document Object Model manipulation, I have provided the output routine for you in the starter code, but you still may want to consider using **console.log()** for additional debugging. In Chrome's console, you can also type in the name of any variable that your script has created and see its contents. You can "drill down" into the Arrays and Objects that you'll be creating in order to make sure that the data is working as expected.

Here is a screenshot of the data you'll be structuring, shown as HTML output on the top and Chrome interactive console data on the bottom. For the console data, after loading the web page, I typed in the names of my two Arrays and then clicked the "reveal triangles" to show the important content. **__proto__** is a built-in property of every Object that we'll learn about in a future week. You can ignore it for now, but you might be able to guess what it does by looking at its output.



1. **TASK:** On Blackboard, download and unzip the files from **Course Materials > Week 2 > Lectures > Workshop 2.** You should end up with a folder containing index.html, workshop2.js, and main.css. Open these files in Bracket or your code editor of choice.

2. **TASK:** Notice that I have created two arrays at the top of **workshop2.js**. One array is used to store all of the Students and their grades. The other is to store Assignments and the total points available on each assignment.

DISCUSSION: As a group, discuss why I might have broken the data up into two arrays instead of creating one many-layered array for this data. Propose alternative structures for storing this data, and discuss the pros and cons. Consider searchability (do we have to loop through the data to find something every time?), clarity (is it easy to understand where the data we need is stored?), and efficiency (have we minimized redundancy of data, while ensuring there's enough overlap between the arrays to cross-reference information?

You might want to refer to the screenshot on the previous page during this discussion in order to see where the data will actually go.

3. **TASK:** You may have come up with a better schema for this data in your group discussion, but for now, let's stick with my plan. It may be a future week's challenge to improve upon my work! First, let's create a Student constructor function. When invoked with **new**, this function should be passed the student's first name and last name as well as a reference to the **gradebookData** array. By a "reference" to **gradebookData**, I simple mean passing that variable to the function so that it knows what array to modify. Remember, arrays are *mutable*!

DISCUSSION: Older languages like C make a distinction between "passing by reference" and "passing by value" while newer languages like JavaScript tend to make assumptions about passing variables around. When you "pass" an Object to a function, you aren't actually making a copy of the Object. You are instead pointing the function at the existing Object and saying "Here, modify this." (remember mutable vs immutable)

This is true for Arrays, Functions, and all other forms of Objects ***except*** for the simple data types: Number, Boolean, and String. When you pass a Number, Boolean, or String into a function, a copy goes in to the function variable, and when the function exits, the copy disappears.

The way I like to think of this is that Objects are too difficult to copy, like trying to Xerox a giant pile of JELL-O, so JavaScript just lazily points to the Object and says "Hey, function, go modify that." Simple data types, on the other hand, are easy to carry around, so JavaScript throws it on the copy machine and hands a copy off to the function, which then throws the copy away when it's done (though the function may lazily point to another Object or throw back a copy of some other simple data in its **return** statement!) Although this was introduced in *Programming Foundations*, it is a tough concept, so if you need a review, please ask for one right now, and we'll do a little demo!

TASK: Within the Student constructor function, create a **firstName** property for the object and set it to the value of the first name that was passed into the function. Then create a **lastName** property for the object and set it to the value of the last name that was passed into the function. Then, create an **id** property for the Student object, and set its value the current length of the **gradebookData** array.

DISCUSSION: Setting the Student ID to the length of the **gradebookData** array ensures that each Student has a unique ID, since the length of an array is always one greater than its highest index (e.g., an Array containing values in positions 0 through 2 will have a length of 3). This works fine for our simple project here in which we are only adding students and grades to the gradebook. If this were an interactive gradebook (in which the user could change values, not just look at them), what actions might mess up this numbering scheme?

4. **TASK:** I have already provided some commented-out test data at the bottom of the JavaScript file, but we aren't ready for that, yet. However, the beauty of the JavaScript Console in Chrome is that we can interactively test the bits of the code that should work. Open the JavaScript Console (option-command-J on your keyboard or **View > Developer > JavaScript Console**) and type in the following:

```
var c = new Student("Charlie", "Chaplin", gradebookData)
```

When you press "return" you will probably get a response of *undefined* because the constructor function does not return an explicit value. If you get an error, then check your work. If not, then just type the letter **c** at the console and hit "return", and you should see the information on your new Student object appear in response! If not, go back and check your code for errors, and ask for help if you need it. This demonstrates how you can interactively test small bits of a JavaScript as you write it.

5. **TASK:** Now that we have a constructor that creates a student, let's write a function to create a new student and add the student to the gradebook all in one step. Call this function **createNewStudent**, and set it up to expect the following information: **firstName, lastName, assignmentData,** and **gradebookData.** The first two pieces of information are Strings, while the last two point to our Arrays. Remember that the information needs to be in the same order in the function header as it is in the function calls that I've already written for you at the bottom of the script!

Now, using the information that's been passed to this function, create a temporary variable (I called mine "student"), call the Student constructor function, and store the results in your temporary variable. Then, using the Array method **push()**, add the student that you've stored in your temporary variable to the end of the **gradebookData** array.

DISCUSSION: Right now we are pushing the new **Student** object on to the **gradebookData** array as soon as it's created, which means that the **Student** object's **id** property will match its array index. Why does this work? What would happen if we didn't immediately add the new **Student** object to **gradebookData** in such a way that other **Students** might get created elsewhere in the script before this one got stored in **gradebookData?** What if instead of pushing the student on to the end of the Array we assigned it directly to the index of the student ID, such as **gradebookData[student.id] = student**? Would that solve the problem?

6. **TASK:** We aren't quite finished with the **createNewStudent** function, but we can test it, even though we haven't done anything with **assignmentData** yet. Functions will happily do nothing with data that they're passed, too! Uncomment the first of my **createNewStudent** function calls at the bottom of the JavaScript, and run the program. You should now see Adam Anders in the HTML output, though his current grade in the class will be **NaN**, which means "Not a Number". **NaN** occurs when you attempt to perform math on non-numeric values. In this case, it's because we've left a lot of grade stuff **undefined**. This may seem like a pain, but it's arguably better than the way some languages like JavaScript used to work, assuming that anything **undefined** was valued as 0. It's better to have a visibly wrong answer like **NaN** than an invisibly wrong one like 0! If you don't see Adam Anders appear in the HTML output, go to the JavaScript console and type in **gradebookData** to see if the object was properly populated with the data (*which should include not only his name and ID, but also an empty array for his grades*)!

7. **TASK:** Now let's do the same for the **createNewAssignment** function. This function will take an assignment name as a String, a total point value for the assignment as a Number, and then a reference to the assignmentData and gradebookData arrays, just as we did for creating a student. The function should then call the constructor function **Assignment** (which I've already written for you) using the data that it's passed, and then it should add the newly created assignment to the end of the **assignmentData** array. It's pretty much the same as what you did in creating a new student, except I wrote the constructor function already! You can test the constructor function in the JavaScript console just as you did for the Student, if you're not sure how it works.

8. **TASK:** Uncomment the second line from my commented-out test data at the bottom of the script and reload in your web browser. You should now see that Homework #1 exists and is worth 10 points, but also that Adam Anders doesn't have a box to be filled in with his homework score!

DISCUSSION: Try examining the **gradebookData** at this point to figure out what's missing!

9. **TASK:** The problem right now is that we have, in database terms, created two tables of data (not to be confused with HTML tables!), but we have not related them to each other. There's nothing in the Student record that says "Hey, there's an assignment out there!" However, we do have that **grades** array that we created for each Student...

This is not a very robust method for connecting data, but it is a good illustration of the power of Arrays. Since we know that we are creating **Assignments**, each with their own unique numeric ID, and since we have a **grades** array that is numerically indexed, why don't we use the numeric indices of the **grades** array to connect our students to our assignments?

Here is what we want to do: each time that we create a new assignment, we want to run a little loop (just three lines of code, including the closing curly brace!) that will iterate through all of the students in our database, and give them a 0 for the new assignment, since they haven't yet done it! You will want to do this inside of **createNewAssignment** after you've created the assignment and stored it, but beyond that, this is your code challenge!

As a hint, if you're iterating through the gradebookData using variable **i**, then you could get to a particular assignment's id for student **i** by looking at **gradebookData[i].grades[assignment.id].** But how do you know which assignment? It's the one you've just created in the **createNewAssignment** function, so you can use your temporary variable that's still storing the **Assignment** object that you created! (In my case, I called it **assignment**, so the above code worked as written, other than needing to assign the 0 for the assignment).

10. **TASK:** Test your program again. If you've done things correctly, then poor Adam Anders should have a 0 on his first homework. If not, then check your work and ask for help!

11. **TASK:** Now uncomment the next two lines at the bottom of the script: the ones for Beth Booker and for Homework #2, and test the program again. Now you should see that Adam Anders has a 0.0 Overall in the class and a 0 on the first two assignments, but Beth has a bigger problem: our **NaN** problem has returned, and furthermore, her Homework #1 is **undefined!**

DISCUSSION: Are these things related? Would this happen for any future students and assignments? Why?

12. The problem is that we have written code to add new Assignment to *existing* Students, but we haven't written code to add all existing Assignments to *new* Students. **TASK:** Add similar code to your **createNewStudent** function so that whenever a new student is added, all assignments currently stored in **assignmentData** are added to the appropriate positions in the new Student's **grades** array. Remember, each Assignment is given an ID in sequence starting with 0, so you can iterate through the Assignments and put them in the appropriate array indices for the new Student.

13. **TASK:** Test the code again. Now you should see two students with two assignments each, and zero points between them! If not, then check your work.

14. **TASK:** For your final challenge, write a function that will update a grade in the gradebook, given the following information: a student ID #, an assignment ID #, the number of points that the student scored, and a reference to our **gradebookData** array. If you write this function successfully (and name it the same as I did), then uncommenting the last two commented lines in **workshop2.js** should result in Adam Anders having 5 points on Homework #1 (for a 16.7% in the class), and Beth Booker having 10 points on Homework #2 (for a slightly better 33.3% in the class).

DISCUSSION: My **updateStudentGrade** function isn't very user-friendly! Imagine if this site were interactive and the teacher had to type in a student ID and an assignment ID in order to update the database! Can you think of ways that we could use the existing data to "look up" a student or assignment by name? Can you think of ways that, once we know how to handle click events in JavaScript, we could perhaps let a teacher update the **gradeBookArray** by just clicking into a table cell and typing in a number? I don't expect you to know how to write this code, yet, but you should be able to imagine how to get to the data that you need, now!

BONUS DISCUSSION: Try doing some math in the JavaScript console. You can just type in something like 24/7 to see what 24 divided by 7 is. If you choose numbers that don't divide into each other evenly (such as 24/7) you should see a big long decimal number appear as the result. However, I managed to format my gradebook results so that the student's overall grade was always a percentage rounded to the nearest tenth of a percent. From the reading this week, can you figure out what built-in Object I used in order to do this? If not, scan through the output code that I wrote for you, and see if you can find it!