**Lesson 6 - Organizing You JavaScript Code:**

| **Title of Unit** | Next Level JavaScript | **Grade Level** | 11-12 |
| --- | --- | --- | --- |
| **Subject** | Mobile App Development | **Time Frame** |  |
| | **Description** | This series digs into the things you need to write larger and larger applications with JavaScript. This is where it gets real! | | --- | --- | | | | |
| **Stage 1 - Identify Desired Results** | | | |
| **Learning Outcomes**  What relevant goals will this unit address? | | | |
| Computer and Information Sciences, General.  **CIP#**: 11.0101  Pathway Competencies   * **Algorithms & Programming**: | | | |
| **Enduring Understandings** | | **Essential Questions** | |
| *Students will understand that…*   * *Understand the basic do’s and don'ts of prototypical inheritance.* * *Understand what computed names and class fields are.* | | *Content specific….* | |
| **Knowledge:** | | **Skills:** | |
| *Students will know how to...*   * *Describe what a prototype is and how it can be used.* * *Explain prototypal inheritance.* * *Explain what Object.create does.* * *Describe common bugs you might run into using constructors.* * *Explain how scope works in JavaScript (bonus points if you can point out what ES6 changed!).* * *Explain what Closure is and how it impacts private functions & variables.* * *Describe how private functions & variables are useful.* * *Explain the module pattern.* * *Describe IIFE. What does it stand for?* * *Briefly explain namespacing and how it’s useful* * *Describe the pros and cons of using classes in JavaScript.* * *Briefly discuss how JavaScript’s object creation differs from a language like Java or Ruby.* * *Explain the differences between using a class to define a constructor and other prototype methods.* * *Explain what “getters” & “setters” are.* * *Describe function binding.* * *Briefly talk about the conflict in JS with functional programming and classes.* * *Explain what npm is and where it was commonly used before being adopted on the frontend.* * *Describe what npm init does and what package.json is.* * *Know how to install packages using npm.* * *Describe what a JavaScript module bundler like webpack is.* * *Explain what the concepts “entry” and “output” mean as relates to webpack.* * *Briefly explain what a development dependency is.* * *Explain what “transpiling code” means and how it relates to frontend development.* * *Briefly describe what a task runner is and how it’s used in frontend development.* * *Describe how to write an npm automation script.* * *Explain one of the main benefits of writing code in modules.* * *Explain “named exports” and “default exports”.* * *Explain the “Single Responsibility Principle”.* * *Briefly explain the additional SOLID principles.* * *Explain what “tightly coupled” objects are and why we want to avoid them.* | | *Students will be able to…*   * *Write an object constructor and instantiate the object.* * *Use inheritance in objects using the factory pattern.* * *Write a factory method that returns an object.* * *Be able to use inheritance with classes.* | |

| **Stage 2 – Assessment Evidence** | | |
| --- | --- | --- |
| **Performance Task** | | |
| **PROJECT: TODO LIST****Introduction** At this point you’ve already had a fair amount of practice using the various techniques we’ve shown you. But we’ve been throwing a *lot* of information your way, so before we move on we’re going to take a minute to slow down and work on one more great portfolio project. **The Todo List** Todo lists are a staple in beginning webdev tutorials because they can be very simple. There is, however, a lot of room for improvement and many features that can be added.  Before diving into the code, take a minute to think about how you are going to want to organize your project | | |
| **Other Evidence** | | **Student Self-Assessment** |
| * Assignment * Practice | | * Reflection |

| **Stage 3 – Learning Plan** | | | | |
| --- | --- | --- | --- | --- |
| * Activity 1 - Organizing Your Javascript Code Introduction * Activity 2- Objects And Object Constructors * Project 6: Library * Activity 3 - Factory Functions And The Module Pattern * Project 7: Tic Tac Toe * Activity 4 - Classes * Activity 5 - Es6 Modules * Activity 6 - Webpack * Project 8: Restaurant Page * Activity 7 - Oop Principles * Project 9: Todo List | | | | |
| **How will you engage students at the beginning of the unit? (motivational set)** | | | | |
| **[Organizing your JavaScript code](https://www.theodinproject.com/paths/full-stack-javascript/courses/javascript/lessons/organizing-your-javascript-code-introduction#organizing-your-javascript-code)** One of the most daunting parts of JavaScript is learning how to organize your code. The reason this subject can be so overwhelming is *not* because JavaScript is so much more complex than other languages, but because it is incredibly forgiving! Many languages force you into using specific patterns and data structures in your code but that is not true in JavaScript.  In the beginning, this is a great thing! For example, if you just want to make a simple button on your webpage do something you can set that up in a couple lines of code. However, as your program becomes more complex, it can become hard to maintain unless you take care to organize your code and because JavaScript is such a flexible language how you do that is entirely up to you. For many coders making decisions about design patterns is crippling so we’re here to help.  This lesson series is going to cover a few of the most common design patterns that occur in modern JavaScript code, we will discuss some pros and cons of each pattern and will give you a chance to practice using each pattern in a project. | | | | |
| **#** | **Lesson Title** | **Lesson Activities** | **CCCs** | **Resources** |
| 1 | Organizing Your Javascript Code Introduction | This lesson series is going to cover a few of the most common design patterns that occur in modern JavaScript code, we will discuss some pros and cons of each pattern and will give you a chance to practice using each pattern in a project.  The patterns we’ll be covering in this series are:   * Plain Old JavaScript Objects and Object Constructors * Factory Functions and the Module Pattern * Classes * ES6 Modules   Going through these will give us a chance to learn about a few other important concepts in JavaScript such as “closure”, “prototypes”, “IIFEs” and more! This series covers the most important parts of JavaScript after simply learning the basics of the language… are you ready? |  |  |
| 2 | Objects And Object Constructors | **Exercise** Write a constructor for making “Book” objects. We will revisit this in the project at the end of this lesson. Your book objects should have the book’s title, author, the number of pages, and whether or not you have read the book  Put a function into the constructor that can report the book info like so:  theHobbit.info() // "The Hobbit by J.R.R. Tolkien, 295 pages, not read yet"  note: it is almost *always* best to return things rather than putting console.log() directly into the function. In this case, return the info string and log it after the function has been called:  console.log(theHobbit.info()); |  |  |
| P6 | Project 6: Library | Assignment  1. If you haven’t already, set up your project with skeleton HTML/CSS and JS files. 2. All of your book objects are going to be stored in a simple array, so add a function to the script (not the constructor) that can take the user's input and store the new book objects into an array. Your code should look something like this: let myLibrary = []; function Book() { // the constructor... } function addBookToLibrary() { // do stuff here } 3. Write a function that loops through the array and displays each book on the page. You can display them in some sort of table, or each on their own “card”. It might help for now to manually add a few books to your array so you can see the display. 4. Add a “NEW BOOK” button that brings up a form allowing users to input the details for the new book: author, title, number of pages, whether it’s been read and anything else you might want. 5. Add a button on each book’s display to remove the book from the library.    1. You will need to associate your DOM elements with the actual book objects in some way. One easy solution is giving them a data-attribute that corresponds to the index of the library array. 6. Add a button on each book’s display to change its read status.    1. To facilitate this you will want to create the function that toggles a book’s read status on your Book prototype instance. 7. Optional -we haven’t learned any techniques for actually storing our data anywhere, so when the user refreshes the page all of their books will disappear! If you want, you are capable of adding some persistence to this library app using one of the following techniques:    1. localStorage ([docs here](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API)) allows you to save data on the user’s computer. The downside here is that the data is ONLY accessible on the computer that it was created on. Even so, it’s pretty handy! Set up a function that saves the whole library array to localStorage every time a new book is created, and another function that looks for that array in localStorage when your app is first loaded. (make sure your app doesn’t crash if the array isn’t there!)    2. Firebase ([check it out!](https://firebase.google.com/docs/?authuser=0)) is an online database that can be set up relatively easily, allowing you to save your data to a server in the cloud! Teaching you how to use it is beyond the scope of this tutorial, but it is almost definitely within your skill set. If you’re interested, check out [this video](https://www.youtube.com/watch?v=noB98K6A0TY) to see what it’s all about. |  |  |
| 3 | Factory Functions And The Module Pattern | **Factory function introduction** The factory function pattern is similar to constructors, but instead of using new to create an object, factory functions simply set up and return the new object when you call the function. **Object Shorthand** A quick note about line 3 from the factory function example. In 2015, a handy new shorthand for creating objects was added into JavaScript. **Scope and Closure** From simply reading the above example, you are probably already in pretty good shape to start using factory functions in your code. Before we get there though, it’s time to do a somewhat deep dive into an incredibly important concept: closure. **Back to Factory Functions** Now that we’ve got the theory out of the way, let’s return to factory functions. Factories are simply plain old JavaScript functions that return objects for us to use in our code. Using factories is a powerful way to organize and contain the code you’re writing. For example, if we’re writing any sort of game, we’re probably going to want objects to describe our players and encapsulate all of the things our players can do (functions!). **Inheritance with factories** In the constructors lesson, we looked fairly deeply into the concept of prototypes and inheritance, or giving our objects access to the methods and properties of another object. There are a few easy ways to accomplish this while using factories. **The Module Pattern** Quick sidenote: ES6 introduced a new feature in JavaScript called ‘modules’. These are essentially a syntax for importing and exporting code between different JavaScript files. They’re very powerful and we WILL be covering them later. They are *not*, however, what we’re talking about here.  Modules are actually very similar to factory functions. The main difference is how they’re created. |  |  |
| P7 | Project 7: Tic Tac Toe | **Assignment**  1. Set up your project with HTML, CSS and Javascript files and get the Git repo all set up. 2. You’re going to store the game board as an array inside of a Game Board object, so start there! Your players are also going to be stored in objects… and you’re probably going to want an object to control the flow of the game itself.    1. Your main goal here is to have as little global code as possible. Try tucking everything away inside of a module or factory. Rule of thumb: if you only ever need ONE of something (gameBoard, displayController), use a module. If you need multiples of something (players!), create them with factories. 3. Set up your HTML and write a JavaScript function that will render the contents of the gameboard array to the webpage (for now you can just manually fill in the array with "X"s and "O"s) 4. Build the functions that allow players to add marks to a specific spot on the board, and then tie it to the DOM, letting players click on the gameboard to place their marker. Don’t forget the logic that keeps players from playing in spots that are already taken!    1. Think carefully about where each bit of logic should reside. Each little piece of functionality should be able to fit in the game, player or game board objects.. but take care to put them in “logical” places. Spending a little time brainstorming here can make your life much easier later! 5. Build the logic that checks for when the game is over! Should check for 3-in-a-row and a tie. 6. Clean up the interface to allow players to put in their names, include a button to start/restart the game and add a display element that congratulates the winning player! 7. Optional - If you’re feeling ambitious create an AI so that a player can play against the computer!    1. Start by just getting the computer to make a random legal move.    2. Once you’ve gotten that, work on making the computer smart. It is possible to create an unbeatable AI using the minimax algorithm (read about it [here](https://en.wikipedia.org/wiki/Minimax), some googling will help you out with this one)    3. If you get this running, definitely come show it off in the chatroom. It’s quite an accomplishment! |  |  |
| 4 | Classes | **Assignment**  1. [This article](https://javascript.info/class) is probably just about all you need to start using class syntax confidently. “Getters and Setters” are a useful feature! 2. [The MDN docs](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes) are, as usual, a great resource for going a little deeper. Look especially at the ‘extends’ and ‘Mixins’ sections. React (and other frameworks) uses classes in this way. You create your components and make them extend the core React component which gives you access to all their built-in functionality. 3. [This article](https://medium.com/@rajaraodv/is-class-in-es6-the-new-bad-part-6c4e6fe1ee65) provides some pros and cons for classes. There are many people who think that class syntax is misleading for Javascript, and thus Factory Functions (from the previous lesson) are inherently better. WE are not saying that classes are bad! We just want you to be informed on the opinions of both sides.  **Practice** Go back to your “Library” example and refactor it to use class instead of plain constructors. |  |  |
| 5 | Es6 Modules | **Npm** The node package manager is a command line tool that gives you access to a gigantic repository of plugins, libraries and tools. If you have done our Fundamentals course, you will probably have encountered it when you installed the Jasmine testing framework to do our exercises.   1. Take a couple minutes to read the [About npm](https://docs.npmjs.com/getting-started/what-is-npm) page - a great introduction to npm. 2. [This video](https://docs.npmjs.com/getting-started/installing-npm-packages-locally) teaches you how to install packages with npm. 3. [This tutorial](https://docs.npmjs.com/getting-started/using-a-package.json) covers the package.json file, which you can use to manage your project’s dependencies 4. If you run into trouble at any point you can check out [the official docs page](https://docs.npmjs.com/) for more tutorials and documentation. |  |  |
| 6 | Webpack | 1. Go through [this tutorial](https://webpack.js.org/guides/asset-management/) to see examples of using webpack to manage your website’s assets. 2. Read through [this tutorial](https://webpack.js.org/guides/output-management/) to learn how to let webpack manually manage your index.html and insert your bundle into the page for you! 3. Finally, the first part of [this tutorial](https://webpack.js.org/guides/development/) talks about source maps, a handy way to track down which source file (index.js, a.js, b.js) an error is coming from when you use webpack to bundle them together. This is essential to debugging bundled code in your browser’s DevTools. If the error comes from b.js the error will reference that file instead of the bundle. It also walks through an example of the --watch feature you *definitely* should have taken note of above. 4. You don’t need to do the rest of the webpack tutorials at this time, but take a look at what’s offered on the sidebar of their [guides page](https://webpack.js.org/guides/). There are several sweet features that you might want to use in future projects such as code-splitting, lazy-loading, and tree-shaking. Now that you have a handle on webpack’s configuration system adding these features is as easy as using the right plugins and loaders! |  |  |
| P8 | Project 8: Restaurant Page | **Assignment**  1. Start the project the same way you began the webpack tutorial project.    1. run npm init in your project directory to generate a package.json file.    2. run npm install webpack webpack-cli --save-dev to install webpack to the node\_modules directory of your project.       1. Quick tip: the node\_modules folder can get *really* big. It is customary to add a .gitignore file to your project so that you don’t have to sync the contents of node\_modules to github. The dependencies that are stored there can be installed from your package.json by running npm install, so you don’t need to sync them.    3. Create a src and dist directory with the following contents:       1. an index.js file in src       2. an index.html file in dist. Go ahead and link the main.js file in a script tag. main.js is the file that will be generated by webpack.       3. create a webpack.config.js file that looks just like our file from the [tutorial](https://webpack.js.org/guides/getting-started/#using-a-configuration). 2. Set up an HTML skeleton inside of dist/index.html with single <div id="content">. 3. Inside of src/index.js write a simple console.log or alert statement and then run npx webpack. Load up dist/index.html in a browser to make sure everything is working correctly.    1. Quick tip #2: if you run npx webpack --watch you will not have to rerun webpack every time you make a change. 4. Create a bare-bones homepage for a restaurant. Include an image, headline, and some copy about how wonderful the restaurant is. It’s okay to hard-code these into the HTML for now just to see how they look on the page. 5. Now remove those elements from the HTML (so leave only the <html>, <body>, and <div id="content"> tags) and instead create them by using JavaScript only, e.g. by appending each new element to div#content once the page is first loaded. Since we’re all set up to write our code in multiple files, let’s write this initial page-load function inside of its own module and then import and call it inside of index.js. 6. Next, set up your restaurant site to use tabbed browsing to access the Contact and Menu pages. [Look at the behavior of this student solution](https://eckben.github.io/bearysBreakfastBar/) for visual inspiration.    1. Put the contents of each ‘tab’ inside of its own module. Each module will export a function that creates a div element, adds the appropriate content and styles to that element and then appends it to the DOM.    2. Write the tab-switching logic inside of index.js. You should have event listeners for each tab that wipes out the current contents and then runs the correct ‘tab module’ to populate it again. 7. If you are using GitHub pages to host your completed page you need to do a tiny bit more work to get it to show up. After running webpack the full bundled version of your site is available in the dist folder, but GH pages is looking for an index.html in the root directory of your project.    1. Simply follow the instructions on [this gist](https://gist.github.com/cobyism/4730490). EZPZ!    2. Recall that the source branch for GitHub Pages is set in your repository’s settings. |  |  |
| 7 | Oop Principles | **Assignment**  1. The following articles mention the acronym SOLID before going on to talk about Single Responsibility. Single Responsibility is definitely the most relevant of the 5. Feel free to dig into the rest of the SOLID principles if you like.. but pay special attention to Single Responsibility.    1. Read [SOLID JavaScript: The Single Responsibility Principle](http://aspiringcraftsman.com/2011/12/08/solid-javascript-single-responsibility-principle/). NOTE: This article does make use of JQuery, one of the earliest and most popular JavaScript libraries prior to the ES6 standard. While The Odin Project does not teach JQuery and you are not expected to understand the example, be sure to focus less on the code itself and more on the SOLID concepts being expressed.    2. [5 Principles that will make you a SOLID JavaScript Developer](https://thefullstack.xyz/solid-javascript/) hits the same topic, and also covers the rest of ‘SOLID’ concisely.    3. And read [S.O.L.I.D. The first 5 principles of Object Oriented Design with JavaScript](https://medium.com/@cramirez92/s-o-l-i-d-the-first-5-priciples-of-object-oriented-design-with-javascript-790f6ac9b9fa) for good measure. 2. [How to Write Highly Scalable and Maintainable JavaScript: Coupling](https://medium.com/@alexcastrounis/how-to-write-highly-scalable-and-maintainable-javascript-coupling-c860787dbdd4) explains loosely coupled objects pretty well. |  |  |
| P9 | Project 9: Todo List | **Assignment**  1. Your ‘todos’ are going to be objects that you’ll want to dynamically create, which means either using factories or constructors/classes to generate them. 2. Brainstorm what kind of properties your todo-items are going to have. At a minimum they should have a title, description, dueDate and priority. You might also want to include notes or even a checklist. 3. Your todo list should have projects or separate lists of todos. When a user first opens the app, there should be some sort of ‘default’ project to which all of their todos are put. Users should be able to create new projects and choose which project their todos go into. 4. You should separate your application logic (i.e. creating new todos, setting todos as complete, changing todo priority etc.) from the DOM-related stuff, so keep all of those things in separate modules. 5. The look of the User Interface is up to you, but it should be able to do the following:    1. view all projects    2. view all todos in each project (probably just the title and duedate.. perhaps changing color for different priorities)    3. expand a single todo to see/edit its details    4. delete a todo 6. For inspiration, check out the following great todo apps. (look at screenshots, watch their introduction videos etc.)    1. [Todoist](https://en.todoist.com/)    2. [Things](https://culturedcode.com/things/)    3. [any.do](https://www.any.do/) 7. Use localStorage to save user’s projects and todos between sessions. 8. Since you are probably already using webpack, adding external libraries from npm is a cinch! You might want to consider using the following useful library in your code:    1. [date-fns](https://github.com/date-fns/date-fns) gives you a bunch of handy functions for formatting and manipulating dates and times. |  |  |

| **Stage 4 - Assess and Reflect** | |
| --- | --- |
| **Considerations** | **Comments** |
| **Is there alignment between outcomes, performance assessment and learning experiences?** |  |
| **Have I made purposeful adjustments to the curriculum content (not outcomes), instructional practices, and/or the learning environment to meet the learning needs and diversities of all my students?** | For struggling students:                    For students who need a challenge: |
| **Do I use a variety of teacher directed and student centered instructional approaches?** |  |
| **Do the students have access to various resources on an ongoing basis?** |  |
| **Have I nurtured and promoted diversity while honoring each child’s identity?** |  |