**Lesson 4 - Etch-a-Sketch:**

| **Title of Unit** | Foundations | **Grade Level** | 11-12 |
| --- | --- | --- | --- |
| **Subject** | Mobile App Development | **Time Frame** |  |
| | **Description** | Developers spend SO MUCH more time reading code than writing it. This is true even with your own code. As a favor to yourself and whoever will need to use, maintain, or further develop your code, please learn to write readable code.  Strings and numbers may be our building blocks but as your scripts get more complex you’re going to need a way to deal with large quantities of them. Luckily, JavaScript has a couple of data types that are used for just that. An Array is simply an ordered collection of items (Strings, numbers, or other things).  Computers don’t get tired, and they’re really really fast! For that reason they are well suited to solving problems that involve doing calculations multiple times. In some cases a computer will be able to repeat a task thousands or even millions of times in just a few short seconds where it might take a human many hours. (obviously speed here depends on the complexity of the calculation and the speed of the computer itself). One way to make a computer do a repetitive task is using a loop.  One of the most unique and useful abilities of JavaScript is its ability to manipulate the DOM. But what is the DOM, and how do we go about changing it? Let’s jump right in… | | --- | --- | | | | |
| **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…*   * *Organizing and storing data is a foundational concept of programming. One way we organize data in real life is by making lists.* * *Each element in an array has a numbered position known as its index. We can access individual items using their index, which is similar to referencing an item in a list based on the item’s position* * *Variables declared with let can be reassigned. Variables declared with the const keyword cannot be reassigned.* | | *Content specific….*   * *What is the DOM?* * *How do you target the nodes you want to work with?* * *How do you create an element in the DOM?* * *How do you add an element to the DOM?* * *How do you remove an element from the DOM?* * *How can you alter an element in the DOM?* * *When adding text to a DOM element, should you use textContent or innerHTML? Why?* * *Where should you include your Javascript tag in your HTML file when working with DOM nodes?* * *How do “events” and “listeners” work?* * *What are three ways to use events in your code?* * *Why are event listeners the preferred way to handle events?* * *What are the benefits of using named functions in your listeners?* * *How do you attach listeners to groups of nodes?* * *What is the difference between the return values of querySelector and querySelectorAll?* * *What does a “nodelist” contain?* * *Explain the difference between “capture” and “bubbling”.* | |
| **Knowledge:** | | **Skills:** | |
| *Students will know how to...*   * *Explain what the DOM is in relation to a webpage.* * *Explain the difference between a “node” and an “element”.* * *Explain how to target nodes with “selectors”.* * *Explain the basic methods for finding/adding/removing and altering DOM nodes.* * *Explain the difference between a “nodelist” and an “array of nodes”.* * *Explain what “bubbling” is and how it works.* * *Syntax for a while loop and a for loop.* | | *Students will be able to…*   * *Create and manipulate arrays* * *Clean their code, or other code to make for a better work environment* * *Create arrays to store a sequence of values in one variable.* * *Access or change elements of an array using “bracket notation”.* * *Add elements to an array using the push() method.* * *Iterate through an array using a for loop.* * *Use loops to repeat code, changing a value in the code sequentially each time (like to draw a row or column of shapes).* * *Nest loops inside each other, a useful technique for changing two dimensions of values (like to draw a grid of shapes).* * *Use the console to debug scripts* | |

| **Stage 2 – Assessment Evidence** | | |
| --- | --- | --- |
| **Performance Task** | | |
| **PROJECT: ETCH-A-SKETCH****Introduction** In this project you’ll be creating a pretty neat toy for your portfolio to flex your DOM manipulation skills. You’re going to build a browser version of something between a sketchpad and an Etch-A-Sketch.  This project should *not* be easy for you. You’ll probably have to Google frequently to get the right JavaScript methods and CSS to use - in fact, that’s the point! You *can* build this using the tools that you have already learned and there are plenty of resources on the net for learning stuff that we haven’t covered yet if you decide you need it. We’ll walk you through the basic steps, but it will be up to you to actually implement them.  If you get totally stuck drop by the chatroom.. someone will be there to point you in the right direction. | | |
| **Other Evidence** | | **Student Self-Assessment** |
| * Practice * Assignment | | * Reflection |

| **Stage 3 – Learning Plan** | | | | |
| --- | --- | --- | --- | --- |
| * Activity 1 - Clean Code * Activity 2 - Javascript Fundamentals Part 4 * Activity 3 - Dom Manipulation * Project 4: Etch-a-sketch | | | | |
| **How will you engage students at the beginning of the unit? (motivational set)** | | | | |
| Developers spend SO MUCH more time reading code than writing it. This is true even with your own code. As a favor to yourself and whoever will need to use, maintain, or further develop your code, please learn to write readable code.  Consider the following 2 snippets of JavaScript:  ugly, hard to read code:  const x = function(z) { let w = 0;z.forEach( function(q){ w += q; });return w; }; x([2, 2, 2]);  clean and easy to read code:  const sumArray = function(array) { let sum = 0; array.forEach(function(number) { sum += number; }); return sum; }; sumArray([2, 2, 2]);  Believe it or not, both of those functions do the exact same thing (in the exact same way!), and both of them are perfectly valid code, but obviously the second one is much easier to follow. Imagine you’re working on a project with someone else and they’ve written the first function… How long is it going to take you to figure out what’s going on there so you can do your work? Imagine you’re working on a project all by yourself and YOU wrote the first function a week or two ago… chances are good that you aren’t going to remember exactly what you were up to there and it’s *still* going to take you a good while to figure it all out again. | | | | |
| **#** | **Lesson Title** | **Lesson Activities** | **CCCs** | **Resources** |
| 1 | Clean Code | **[Assignment](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/clean-code#assignment)** Read through these articles that discuss a few elements of writing good clean code.   1. [This list of clean-code tips](https://onextrapixel.com/10-principles-for-keeping-your-programming-code-clean/). 2. [This article](https://blog.codinghorror.com/coding-without-comments/), [and this one too](https://blog.codinghorror.com/code-tells-you-how-comments-tell-you-why/) about the role of comments in your code. |  |  |
| 2 | Javascript Fundamentals Part 4 | **Practice Test Driven Development** Test Driven Development (TDD) is a phrase you often hear in the dev world. It refers to the practice of writing automated tests that describe how your code should work before you actually write the code. For example, if you want to write a function that adds a couple of numbers, you would first write a test that uses the function and supplies the expected output. Before you write your code the test will fail, and you should be able to know that your code works correctly when the tests pass.  In many ways TDD is much more productive than writing code without tests. If we didn’t have the test for the adding function above, we would have to run the code ourselves over and over, plugging in different numbers until we were sure that it was working… not a big deal for a simple add(2, 2), but imagine having to do that for more complicated functions, like checking whether or not someone has won a game of tic tac toe: (game\_win(["o", null,"x",null,"x",null,"x", "o", "o"])) If you didn’t do TDD then you might actually have to play multiple games against yourself just to test if the function was working correctly!  We will teach you the art of actually writing these tests later in the course. The following exercises have the tests already written out for you. All you have to do is read the specs and write the code that makes them pass! The very first exercise (01-helloWorld) is intentionally very simple and walks you through the process of running the tests and making them pass. **Good Luck!** Check out our exercises repository [here](https://github.com/TheOdinProject/javascript-exercises) and follow the directions in the [README](https://github.com/TheOdinProject/javascript-exercises#how-to-use-these-exercises) for setting up Jest. Solutions for the exercises can be found on the ‘solutions’ branch of that repo.  Complete the following exercises:   * helloWorld * repeatString * reverseString * removeFromArray * sumAll * leapYears * tempConversion |  |  |
| 3 | Dom Manipulation | **Exercise** Copy the example above into files on your own computer. To make it work you’ll need to supply the rest of the HTML skeleton and either link your javascript file, or put the javascript into a script tag on the page. Make sure everything is working before moving on!  Add the following elements to the container using ONLY javascript and the DOM methods shown above.   1. a <p> with red text that says “Hey I’m red!” 2. an <h3> with blue text that says “I’m a blue h3!” 3. a <div> with a black border and pink background color with the following elements inside of it:    1. another <h1> that says “I’m in a div”    2. a <p> that says “ME TOO!”    3. Hint for this one: after creating the div with createElement, append the <h1> and <p> to it before adding it to the container.  **Practice** Manipulating web pages is the primary benefit of the JavaScript language! These techniques are things that you are likely to be messing with *every day* as a front-end developer, so let’s practice!   1. Do the first exercise in Wes Bos’s JavaScript30 program by cloning the repo at <https://github.com/wesbos/JavaScript30>. Check out the [Video Tutorial](https://www.youtube.com/watch?v=VuN8qwZoego) for instructions on the project. 2. Go back to your “Rock Paper Scissors” game from a previous lesson and give it a simple UI so that the player can play the game by clicking on buttons rather than typing their answer in a prompt.    1. Copy your original code into a new file so you don’t lose it.    2. For now, remove the logic that plays exactly five rounds.    3. Create three buttons, one for each selection. Add an event listener to the buttons that calls your playRound function with the correct playerSelection every time a button is clicked. (you can keep the console.logs for this step)    4. Add a div for displaying results and change all of your console.logs into DOM methods.    5. Display the running score, and announce a winner of the game once one player reaches 5 points.    6. You will likely have to refactor (rework/rewrite) your original code to make it work for this. That’s OK! Reworking old code is an important part of the programmer’s life.    7. Don’t forget to go back and add your new version to the original Rock Paper Scissors Project! 3. Watch the [Event Capture, Propagation and Bubbling video](https://www.youtube.com/watch?v=F1anRyL37lE) from Wes Bos’s JavaScript30 program. If you want to code along with the video, you can use the contents of folder #25 from the repo you cloned above. |  |  |
| P4 | Project 4: Etch-a-sketch | **Assignment**  1. Setup a git repository for this project. 2. Create a webpage with a 16x16 grid of square divs    1. Create the divs using JavaScript… don’t try making them by hand with copy and pasting in your html file!    2. Best to put your grid squares inside another “container” div (that one can go directly in your html)    3. There are several different ways to make the divs appear as a grid (versus just one on each line) feel free to use any or play with each of them:       1. float/clear       2. inline-block       3. flexbox       4. CSS Grid    4. Be careful with borders and margins, they can adjust the size of the squares!    5. “OMG, Why isn’t my grid being created???”       1. Did you link your CSS stylesheet?       2. Open your browser’s developer tools       3. Check if there are any errors in the JavaScript console       4. Check your “elements” pane to see if the elements have actually shown up but are somehow hidden.       5. Go willy-nilly and add console.log statements in your JavaScript to see if it’s actually being loaded. 3. Set up a “hover” effect so that the grid divs change color when your mouse passes over them, leaving a (pixelated) trail through your grid like a pen would.    1. Hint: “hovering” is what happens when your mouse enters a div and ends when your mouse leaves it.. you can set up event listeners for either of those events as a starting point.    2. There are multiple ways to change the color of the divs, including:       1. adding a new class to the div       2. changing the div’s background color using JavaScript. 4. Add a button to the top of the screen which will clear the current grid and send the user a popup asking for how many squares per side to make the new grid. Once entered the new grid should be generated *in the same total space as before* (e.g. 960px wide) and now you’ve got a new sketch pad. Tip: Set the limit for the user input to a maximum of 100. A larger number of squares results in more computer resources being used, resulting in possible delays, freezing, or crashing that we want to prevent.    1. Research button tags in HTML and how you can make a JavaScript function run when one is clicked.    2. Also check out prompts    3. You should be able to enter 64 and have a brand new 64x64 grid pop up without changing the total amount of pixels used 5. (Optional): Instead of just changing the color of your grid from black to white (for example) , have each pass through it with the mouse change to a completely random RGB value. Then try having each pass just add another 10% of black to it so that only after 10 passes is the square completely black. 6. Push your project to GitHub! |  |  |

| **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:      Students sometimes forget that they actually have to *use* the loop counter variable in their shape commands - otherwise their program will just draw a bunch of shapes on top of each other.   * Students may wonder when they should use a while loop versus a for loop. A while loop is the most general kind of loop, a for loop is a more specific syntax that’s great for iterating through a sequence of numbers. For most drawings and animations, students will want to use a for loop. * Students often forget all the parts of the for loop header. They can remember it by checking out the [documentation example](https://www.khanacademy.org/computer-programming/for-var-i-0-i-8-i-1/877960284). Remind them that they don’t have to memorize programming syntax at first, as they will often have documentation - the important thing is to practice using the syntax and to understand it. * Students may write loops with the classic "off by one" error. That’s when their condition isn’t quite right, because they’re stopping one iteration too early or too late. Encourage them to think carefully about their condition, and what the values will be in the final iteration of code that will be executed. * Students take time to get used to the fact that arrays start at index 0, since we’re used to counting from 1 in our every day life. Keep reminding them about that. * If students didn’t get comfortable using for loops in the last lesson, they will likely struggle to use them in this lesson. If that’s the case, they may want to re-watch the for loop lesson and go through the challenge again. * When iterating with a for loop, students often forget to use bracket notation to access the elements of their array, and they instead access the whole array. Ask them whether their code is truly accessing only one element of the array. * Students may wonder if there is a method for removing elements from an array. They can use the [pop()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop) method for that. There are in fact many array methods we don’t teach in the intro course, which students can find by searching the web for “JS array methods”. * Students may wonder if they can create a 2-d array, to store a grid of values. They can certainly do that, but we don't teach that in the intro course. Search for “multi-dimensional JS array” or “2-d JS array” for examples.     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?** | **Additional materials: Discussion questions**  These are questions that you can ask students individually after they've done the lesson, or lead a group discussion around, if everyone's gotten to the same point.   * What data would be useful to store in an array? Open a popular game or an app for inspiration, and have the students brainstorm.   These are questions that you can ask students individually after they've done the lesson, or lead a group discussion around, if everyone's gotten to the same point.   * Look back at previous programs your students have made. Find ones with rows/columns of shapes, and ask students to point out where they could use loops. Perhaps if they have time, they can upgrade them. * Look around the room. If you were drawing the room, where would you use loops?   An "unplugged" activity is one that you can do with students without needing to use a computer at all. They can help convey concepts in a more visceral way, and they can also be a backup activity for when computers fail.  Code.org offers a For Loops Fun activity (see [overview video](https://www.youtube.com/watch?v=mX857v5B4gE), [lesson plan](https://code.org/curriculum/course4/8/Teacher), and [example in-class video](https://www.youtube.com/watch?v=HO4uhoahtjM&list=PLzdnOPI1iJNerXmhWGR_V-8vWPe0v62DE&index=15)). **Additional materials: Trivia questions** These can be fun to do as a class after everyone’s gotten through the lesson. They can also lead to discussion about which questions are the hardest. Play them on [Quizizz](http://quizizz.com/admin/quiz/56f48502bb00b31b0a1349c2)  These can be fun to do as a class after everyone’s gotten through the lesson. They can also lead to discussion about which questions are the hardest. Play them on [Quizizz](http://quizizz.com/admin/quiz/56f31ce567f3c34f43c31314). |
| **Have I nurtured and promoted diversity while honoring each child’s identity?** |  |