50 States Guessing Game

In this module, we are going to create a game where the user needs to guess the 50 states in america to get to the next page. We are going to use the concepts you learned in the previous module (variables, functions, and conditional logic) and we are going to also introduce arrays! We will once again specify the controls for our page in HTML, add our functionality in JavaScript, and styling our page in CSS.

Creating our files

Go to 'guessing-game.html'., which you created in the login step, and set up your code structure just as you did in 'index.html';

<html> (every html file must start and end with these tags)

<head>

(the head is like the brain of your file. I'll show you what goes here in a bit!)

</head>

<body>

(This is where your content goes that you see on your site.)

</body>

</html>

Create an empty **'game.js'** javascript file, and save it in your **'js'** folder next to **'login.js'**.

Reference both your 'style.css' and your 'game.js' files between your <head></head> tags.

```
<head>
    <link rel="stylesheet" type="text/css" href="css/style.css">
    <script src="js/game.js"></script>
</head>
```

In between the <body></body> tags, add a div with the same class as the login page ‘main’. Remember we created this class for our login page? We can use this class for all of our containing divs to keep the style the same on each page.

Your page <body> </body> should l look like this:

```
<body>

    <div class="main">

    </div>

</body>
```

Now your page is set up, with your CSS and JavaScript already referenced!

How The Guessing Game Works

In this guessing game, we are asking the user to guess all 50 states in America before they can move to the next page. The user enters their guess into a textbox, and the JavaScript checks if this is a state.

If the guess is correct then the name of the state goes into a ‘correct answers’ panel, and a message shows to tell them how many states are left to guess.

If the state is incorrect, nothing is added to the correct answers panel and the count of how many are left remains the same.

Just as we did with our login screen, let’s think about what controls we might want on the Guessing Game page

- Text telling the user how to play the game
- A textbox for them to type their guess
- Another, larger text box which shows the user a list of their correct guesses.
- A label to show the count of states that are left to guess

Copy this code in between the tags in your page body, and then we can go through each control

```
<h2> Can you guess all of the states in America?</h2>

 <p> Enter your guesses below to make it to the next page:</p>

 <input class="guessingGame" type="text" id="userGuess" onkeyup="guess()">

 <p>Correctly guessed:</p>

 <textarea class="correctAnswers" id="correct" disabled="true"></textarea>

 <label id="counter"></label>
```

As you can see, this is a very simple interface!

The first two lines show text to the user using <h2> and <p> tags.

- A **<h2>** tag is a header tag, and is usually used to show important information on the page as the font is big and bold. Text within a <h2> tag is pre-styled so it's very helpful to use the <h2> tags as you can change the size of text without having to change the css. You can also use <h1> tags for even larger text. Click here to read about Header tags.
- A **<p>** tag places the content within it into its own separate paragraph. This text is not formatted, and is the default style for your website.

Below this, we have a text box which is set up almost exactly the same as the login page, with styling, type, and ID attributes. This textbox however has an extra attribute, called 'onkeyup' , and this is where our JavaScript function task will be kicked off.

Event Attributes

'onkeyup' is an event that is fired in your browser whenever someone types something into a textbox. What we are doing here is saying: when the 'onkeyup' event occurs (when the user types something), I want the browser to start the **guess**() function in our JavaScript file. This function will then see if the text that the user has typed matches the name of a state.

There are a lot of event attributes that you can add functions to in your HTML. The most common one is an OnClick event which is added to a button element, and controls what happens when the button is clicked. However there are many more! Here is a list of all of the event attributes. Events are a way to trigger functions in your JavaScript and make something happen. Clicking a button, checking a checkbox, choosing something from a dropdown are all events that you can attach to a JavaScript function through an event attribute.

Next we have another <p> tag with some text, and then a <textarea></textarea> where we will put the correct guesses. A text area is larger than a textbox, so it's more suitable for this. You can see we have added the following attributes:

- **Class**: So it can be styled in style.css.
- **ID**: So we can reference the control in the JavaScript to show correct guesses.
- **Disabled:** We don't want the user typing anything in here.

Question: Finally, we have a label to show the number of guesses left. As you can see, this label has an ID called "counter", whereas the labels in your login page did not. **Can you guess why this is?**

Answer: It's because we want to reference this label in the JavaScript so that we can tell it how many guesses are left and what text to display. We didn't want to reference the labels on our login page, they were just for displaying information.

Question: From this code, you can see which controls we will be referencing in the JavaScript and which we won't by seeing which ones have an ID. Do you remember the JavaScript from our login page that finds an element from it's ID?

Answer: document.getElementById("elementID")

Refresh your page and you can see our elements here. Like the login page, we will finish styling this later, but you can already see that the div has the 'main' class, so our content is in the middle of our page with a grey border.

This is a great benefit of CSS, and means we don't have to write any more code to get the content in position. We just need to apply existing classes to our elements on the new page.

Adding our JavaScript

In our login page, I introduced you to variables, functions, and conditional statements. Here I will use all three of these, along with a new concept called an array.

Think of an array as a variable that stores a list of data. A variable can store one name

Var name = “Peter”;

But an array can store a list of names:

Var nameArray = [“Peter”, “Mary”, “John”];

You can then access all of the items in your array by calling this array, rather than having to call multiple variables. It makes it much quicker to write code that deals with lists of data. For 50 states we don’t need 50 variables, we just need 1 array!

Extra reading: [Here is a useful article about arrays.]

Open your game.js file, and let’s start writing our code!

As with the login.js file, we’re going to start by declaring our variables - copy this code into the top of your game.js file:

```
var numOfGuesses = 50;

var correctlyGuessed = 0;

var correctAnswersArray = [];

var statesArray = ['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
'Colorado', 'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',
'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana',
'Nebraska', 'Nevada', 'New Hampshire', 'New Jersey', 'New Mexico', 'New York', 'North
Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode
Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont',
'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyoming']
```

This sets up the following:

- **numOfGuesses**: This is the count of how many guesses that the user has.
- **correctlyGuessed**: This keeps a count of how many guesses are correct. We will add 1 each time the user enters a correct guess.
- **correctAnswersArray:** We are creating an empty array which will store a list of the correct guesses, which will be outputted in the 'correctAnswers' textarea on the web page.
- **statesArray:** This is an array which is filled with a list of the states in america.

Next, let's create our Guess() function, which is the task that is kicked off when the user types something into our textbox.

Copy the following code into your game.js file, just below where you created your variables.

```
function guess() {

    var userGuess = (document.getElementById("userGuess").value);

    if (statesArray.includes(userGuess) && !correctAnswersArray.includes(userGuess)) {

        correctAnswersArray.push(userGuess);

        document.getElementById("correct").innerText = correctAnswersArray.toString();

        document.getElementById("userGuess").value = " ";

        correctlyGuessed++;

        var guessesLeft = numOfGuesses - correctlyGuessed;

        document.getElementById("counter").innerText = (guessesLeft) + " guesses left.";

    }

}
```

The ‘Guess’ Function

As we defined in the HTML, the ‘Guess’ function is called whenever a user types something into the textbox. So each time a letter is typed in here, the function checks the entire content of the textbox and sees if it matches anything that it is in our **statesArray.**

The first thing we do in our function is get the contents of the userGuess textbox and assign this to a variable: ‘userGuess’. We do this in the exact same way we did with the username and password in our login screen.

After we’ve created the variable, we have a conditional ‘if’ statement like we used in the login code, which checks if the contents of the textbox match any of the items in the list of states (stateArray).

Checking an array of 50 items for 1 match seems like it would require a lot of code, but luckily JavaScript provides a function for doing this in just one word: **‘includes’.**

Javascript **includes**() is an inbuilt function that determines whether the array contains a specified element or not. It returns true or false as the output depending on whether the array contains the item or not. It’s a necessary function to know if you’re working with arrays, as it saves SO MUCH time!

Here is what our if statement is doing:

- **If(statesArray.includes(userGuess) -** if the states array contains a match for the userGuess variable
- **&&** - a double ampersand means that there is a second condition which must be true
- **!correctAnswersArray.includes(userGuess)** - If the correctAnswersArray **does not** contain the guess variable. (Putting an exclamation point in front of this code means that we want to check that the statement is not true.)

Here, we are saying if our guess is correct (in the statesArray) AND if it hasn’t been guessed before (is not in the correctAnswersArray), then you can proceed with the next bit of code. There is no ‘else’ statement, as we will do nothing if the guess isn’t correct.

Question: If I were to write a statement that says “if a person is male, and his name isn’t Paul, then run this task” - how would I write that as a conditional statement? Practice and then look at the answer below to compare your code!

Answer:

```
If (person == “male” &&  !name == “Paul”) {

        //run this piece of code

}
```

What to do if the guess is correct

Within the conditional statement is the code for what to do if the guess is correct. Here is what this code does:

1. **Adds the guess to the correctAnswersArray.**

   We have an array called correctAnswersArraywhich keeps track of how many guesses are correct. We need to add the value of our variable **'userGuess'** to this array. To do this, we use the **push**() function.

   This takes the following syntax: *arrayname.push(dataToAdd);*

2. **Adds the guess to the text area on the page.**

   We can set the text of the 'correct' text area using **document.getElementById("correct").innerText.**

   We will do this using the **'ToString()'** function which is another inbuilt JavaScript function that you can use on arrays.. This function turns an array of data into a list of items separated by a comma, so that it can be placed into the textbox. If you do not add the ToString() function, you won't be able to place the array contents directly into the box as it will be treated like a JavaScript object and not a string of text.

3. **Empties the guessing textbox.**

   As you can see, the code sets the value of the 'userGuess' textbox to be **" "** - an empty space.

4. **Increases the correctly guessed count by 1.**

   Remember in login.js, when we decreased the number of attempts with 'attempts--'? Now we are doing the opposite. We are adding 1 to our variable instead.

   -- means take one away from the existing value and ++ means add one to the existing value.

   You can also say correctlyGuessed = correctlyGuessed + 1; but this requires more code. So we say correctlyGuessed++ instead.

5. **Calculates the number of guesses left**

   Here we are creating a new variable called guessesLeft, where we are taking the value of correctlyGuessed from numOfGuesses. This gives us the number of guesses remaining so that we can show the user on the page.

6. **Updates the webpage to show the amount of states left to guess.**

   Lastly, set the innerText of your "counter" label to show your total guesses left. As you can see from this line of code, if you want to display a variable value and static text together in your label, you need to put a '+' between them. So **control.text = variable + "some kind of text";**

   Try removing the '+'. You will see a red line under the text to show that there is an error in your code.

Testing your code

Save this code, and open guessing-game.html in your browser. Type **'Delaware'** into the textbox, and see that the code works as expected. Now type 'delaware', with a small 'd'. Does this register as a correct guess?

The answer is no, and that is because the **array.Includes()** function that we used in the code is case sensitive. So what is typed into the textbox needs to be *exactly* the same as what's in the array - uppercase and lowercase guesses won't match. That's not ideal, is it? People may type Delaware / delaware / DELAWARE and the game should accept all three.

Not to worry though, there is a way to fix this!

How to handle case sensitivity

When you're building a website, case sensitivity is a really important thing to check for. You will often be performing checks on user inputted data, so you must make sure that these checks do not depend on the user using a certain case.

The easiest way to do this, is to use a JavaScript built in function called **.toUpperCase()** and convert both the correct answer and the guess to be uppercase in the code before comparing them. This way you are always comparing like for like, and case sensitivity will not be a problem.

The first thing we need to do here is convert every item in our **'statesArray'** to be uppercase. You might think that converting all 50 items will take a lot of code, but JavaScript provides a built **.map()** function that makes this super easy.

This function lets you take an array and make a new array from it's values after performing an action on these values. So you can say "take all of the values in my array, make them uppercase and map them to a new array that I can use in my code."

The .map() function means you can do all of this in three lines of code!

```
var upperCaseStates = statesArray.map(function(value) {

    return value.toUpperCase();

  });
```

Here we are creating a new variable array called **'upperCaseStates'** and we are mapping all of the values in '**statesArray'** to this new array, after we have called the .toUpperCase(); method on them.

Click here to learn more about the .map() function.

Add that piece of code above your 'guesses()) function, just below where you declared your variables.

Task

See if you can change the Guesses() function so that you are searching your new uppercase array for the userGuess in uppercase.

Right now you are searching your regular statesArray for the text exactly as it is typed in by the user. Can you change the code to use the new uppercase array and use the .toUppercase() function to make the userGuesses variable case insensitive?

Answers are below, but try this without checking the answer and compare your code.

Answer

```
function guess() {
  var userGuess = (document.getElementById("userGuess").value).trim().toUpperCase();
  if (upperCaseStates.includes(userGuess ) && !correctAnswersArray.includes(userGuess))
    {
      correctAnswersArray.push(userGuess );
      document.getElementById("correct").innerText = correctAnswersArray.toString();
      document.getElementById("userGuess").value = " ";
      correctlyGuessed++;
      var guessesLeft = numOfGuesses - correctlyGuessed;
      document.getElementById("counter").innerText = (guessesLeft) + " guesses left." ;
    }
 }
```

Here, we are adding two extra ‘methods’ - .trim() and .toUpperCase() to our guess:

- **.trim():** This removes any extra spaces in the text. So if the user typed “Delaware “ with that space at the end, it wouldn’t exactly match what is in the ‘statesArray’ array and would not be taken as a correct guess.
- **.ToUpperCase()**: This converts anything the user has typed into uppercase, so we are checking for the right case.

Testing

Test out the code by typing out states in a mix of cases and see if it works! **Hurrah! You did it!!!!**

Troubleshooting

If your game is not working, go through the above code and make sure it's all exactly matching what is in this document.

Check that your HTML file has been referenced your JavaScript and CSS file in the <head</head> section.

In your browser, click 'CTRL + Shift + I', or F12 which opens a panel at the bottom or side of the screen. This is the developer panel, and it's where your browser will show any JavaScript errors you have made. Go to the 'console' tab and any red lines in here will be errors. You can see the line number that has the error and check the code on that link.

As a last resort, go to the completed project and compare your code with the code in this project!

Styling your Guessing Game

Your guessing-game.html file references your style.css file, so you can add new classes in this file which can be added to the elements in your HTML file using the attribute "class".

Add the following classes to your css file at the end.

```
 .guessingGame {
      width: 100%;
      height: 25px;
  }
  .correctAnswers {
    width: 100%;
    min-height: 125px;
    margin-bottom: 20px;
}
.guessingGameBody {
    background: url("../img/american-flag.jpg") 50% 50% / 100% no-repeat fixed;
}
```

Add the class **'guessingGame'** to your text box, **'correctAnswers'** to your textarea, and '**guessingGameBody'** to your page <body> using the class attribute. Try to do it without looking at the below code:

```
<body class="guessingGameBody">
```

```
<input class="guessingGame" type="text" id="userGuess" onkeyup="guess()">
```

```
<textarea class="correctAnswers" id="correct" disabled="true"></textarea>
```

Refresh the page, and see how this changes the page. We have styled our elements, and added a flag as the page background!

Tasks

I want you to practice your CSS by styling this page.

- Using the background-color property, change the background color of the box that contains the controls.
- Add another american flag above the <h2> tags, like we did on the login page with our avatar.
- Make the box that the elements are within bigger, without changing how things look in the login page.
- Change the font colours to red, white, and a blue

Next, practice your JavaScript to change the game functionality. This is advanced, and the answers are below - but I want you to give it a shot and see how much you have learned.

- Add a conditional statement within the **'guesses**()' function to check if the user has guessed all of the correct answers.
- If this condition is true, show a popup message (like we do in the login page) that says '**Congratulations, you have made it to the next round**!',
- Then send the user to the next page using the **window.location** object. (The page I have linked will help you to do this.) The next page we will be creating will be 'math.html', so create this page in your project folder and direct users to the page.

See if you can change the code to add this extra functionality!

Helpful hint: Rather than guessing all 50 guesses to see if you reach this condition, change the numOfGuesses variable to be '3' instead of 50 and the condition will be reached after 3 guesses.

Answers

Styling

- Add ‘background-color’ to the ‘main’ class if you want to add the background color here.
- Add the image in the same way as you added your image to the login page. Just find a new image online and replace the image reference in the <img> tag in your html.
- Add a class to your elements which has the property ‘color’. This sets the font colour of text within the element.

Functionality

In the line below **‘correctlyGuessed**++’ in your Javascript, add your conditional logic to check if your correctly guessed answers equals your **numOfGuesses** (50).

In here, create your alert with your message, and then change your window location to be the next page.

```
if (correctlyGuessed == numOfGuesses)
        {
            alert("Congratulations, you have made it to the next round!");
            window.location = "math.html";
        }
```

Did you get this right? If not, take a look through the code and see if you can add some more conditions that could be met.

Perhaps if the user reaches 10 guesses it can show them an alert to tell them how many they have left?

Or try another challenge - see if you can change the list of items that are being guessed. Swap out states for movies, presidents, counties in Ireland (change the background image to an Irish flag). See what you can do here with the knowledge and skills that you have just learned.

What we have learned so far

- What an array is.
- How to search an array for an item.
- How to convert an array of data to a string of text to display on a webpage.
- How to add an item to an array.
- How to take an array, perform a function on the data, and add that data to a new array.
- How to call a Javascript function on a keyboard event.
- How to go to the next page in Javascript.