# Use JavaScript to Create a "Code Breaker" Game

You'll build a Code Breaker game using JavaScript. Based off the board game Mastermind, the game will randomly generate a hidden code and the player gets 10 attempts to guess that code based on provided feedback.

# Live Demo

<https://codeschool-projects.github.io/CodeBreakerProject/> to see a working version of this project.

You'll build a Code Breaker game that you can play and show off to others as an example of your abilities in JavaScript.

# Setup Instructions

In this project, all of your changes will happen in the `/src/assets/main.js` file.

# Tasks

Complete the following tasks to finish this project.

## Create `setHiddenFields` function

Create a function named `setHiddenFields` that sets the `answer` variable equal to a randomly generated whole number between 0 and 9999.

\*\*Hint:\*\* `Math.random()` can be used to randomly generate a number between 0 and 1 (up to 18 decimal points) and `Math.floor(input)` can be used to round down to the nearest whole number.

## Make sure the hidden input `answer`'s value is exactly 4 characters long

In our `setHiddenFields` function we need to make sure the hidden input `answer` is exactly 4 characters long. (If our random number generates "42", we want to set the `value` of `answer` to "0042".)

\*\*Hint:\*\* In order to add a zero to the front of an answer, it must be a string, not a number. You can convert numbers to strings with `.toString()`. We can create a `while` loop that runs while `answer.length` is less than 4 that puts a `0` before answer's current value.

## Set the hidden input `attempt`'s value to zero

In our `setHiddenFields` function, we should also set the hidden input `attempt` to `0`.

## Only set the `answer` and `attempt` hidden inputs when they aren't already set

Call the `setHiddenFields` function in the body of the `guess` function, but also write some logic so that it's only called when answer and attempt haven't already been set.

\*\*Hint:\*\* we can use an `if` condition to only run our code when `answer` or `attempt` is empty (`''`).

## Create `setMessage` function

Create a `setMessage` function with one parameter. This function should set the `message` label to whatever is provided to the parameter.

\*\*Hint:\*\* With a label, you'll want to set its `.innerHTML`, not its `.value`.

## Create `validateInput` function

Create a function `validateInput` with one parameter. If the parameter has a `length` of 4, return `true` â€” otherwise, use the `setMessage` function to set the `message` label to `"Guesses must be exactly 4 characters long."`, then return `false`.

## Call the `validateInput` function from the `guess` function

Create an `if` condition block that uses `validateInput` with a parameter of `input.value` as the conditional. If `validateInput` returns `false`, then use `return false` to stop execution of the `guess` function, otherwise we should increment the `attempt` hidden input by 1.

\*\*Hint:\*\* You can negate a value on the `if` statement by using the exclamation point, like this: `if(!someValue)`.

## Create `getResults` function

Create a `getResults` function that has one parameter. In this function, we need to add the results of the user's guess to our `results` div's `innerHTML`. Each result should begin with `<div class="row"><span class="col-md-6">' + input + '</span><div class="col-md-6">` where `input` is the value the user guessed. Then for each character, you should add `<span class="glyphicon glyphicon-ok"></span>` if the character is in the correct position in the `answer`, a `<span class="glyphicon glyphicon-transfer"></span>` if the character is in the `answer` but isn't in the right position, and `<span class="glyphicon glyphicon-remove"></span>` if the number isn't in the `answer` at all. Don't forget to close your divs!

\*\*Hint:\*\*\* You can create a variable to hold the initial div, then add each character's results to that variable in a `for` loop, then add the closing `div` tags after the loop. After which you can just set the `results` element's `innerHTML` to that variable.

## Check for correct guess

In our `getResults` function, create a variable that counts how many characters were guessed correctly. If all characters were guessed correctly, the function should return `true`, otherwise `false`.

## Set up win condition

Add a call to the `getResults` function at the end of our `guess` function. If `getResults` returns true, use the `setMessage` function to set the `message` label to `"You Win! :)"`.

## Set up lose condition

If `getResults` returns `false` and the hidden input `attempt` value is greater than or equal to 10, use the `setMessage` function to set the `message` label to `"You Lose! :("`.

## Continue play condition

If neither a win or lose condition is met, use the `setMessage` function to set the `message` label to `"Incorrect, try again."`.

## Create a `showAnswer` function

Create a function `showAnswer` that has one parameter. This function should set the `innerHTML` of the `code` label to the `value` of the `answer` hidden input. In addition to this, it should take the parameter as a `true` or `false` (indicating if the player won or lost) if the parameter is `true` add ` success` to `code`'s `className` otherwise, it should add ` failure`. (Note the space before ` success` and ` failure`.)

## Create a `showReplay` function

Create a function `showReplay` with no parameters. This function will change the `style.display` of `guessing-div` div to `none` and the `style.display` of the `replay-div` div to `block`, making it so the user can start over after they win or lose the game.

## Add `showAnswer` and `showReplay` to win / lose conditions

When a player wins in addition to `setMessage` call, they should also call `showAnswer` passing `true` for its parameter, and finally make a call to `showReplay`. When the player loses, they should call `showAnswer` with `false` for the parameter and then `showReplay`.