**Step 3 of 6**

**Accept & Validate Player Guesses**

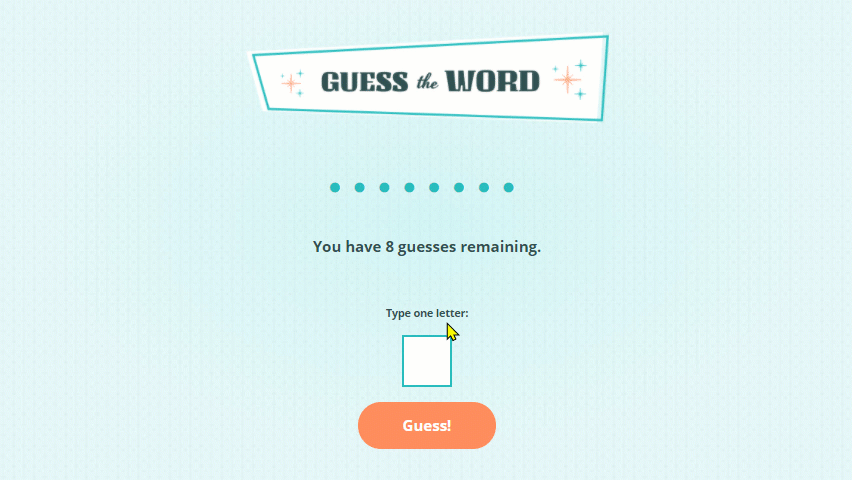
In this part of the project, you’ll write two new functions and validate the player’s input to make sure they only input a single letter.

The first function you’ll create checks if the player’s input is a letter. To do this, you’ll use a regular expression. A **regular expression** lets you find text that matches a specific pattern, like alphabetic text only (no number or symbols). Because this is your first time working with a regular expression, we’ll give you the line of code you’ll need to make sure your player only inputs letters. If you want to read more about regular expressions, check out [MDN’s Regular Expressions(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\_expressions)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions) page.

There’s one other new piece of code you’ll work with—match(). The **match() method** works with the regular expression to search the strings (i.e., the letter the player inputs) to match it to the regular expression. We’ll give you a hint on where to use this method in your project.

You’ll then return to the click event you made in the last step to validate the player’s input to make sure they’ve only added a single letter. Finally, you’ll write a second function that captures the player’s guess to see if they’ve already guessed that letter. If not, the function pushes the letter to an array of guessed letters.

When finished, you’ll test your game to make sure that the game only accepts single letters as a guess. You’ll also ensure that a message appears on the screen if the user guesses the same letter more than once.



*When a player guesses the same letter twice or enters a non-alphabetic character, a message appears to let them know they already guessed the letter or enter a letter from A to Z.*

**What to Do:**

**Create a Function to Check Player’s Input**

1. Create and name a function that accepts the input value as a parameter. This function’s purpose is to validate the player’s input.
2. Inside the function, create a variable for the accepted letter sequence: const acceptedLetter = /[a-zA-Z]/. Now your code uses a regular expression to ensure the player inputs a letter!
3. Still inside the function, use a conditional block to check for different scenarios. First, check if the input is empty. Then, check if the player has entered more than one letter. Finally, check if they’ve entered a character that *doesn’t match* the regular expression pattern. Hint: You’ll need the .match() method here. Each condition should have a message directing the player on what to input.
4. If all the other conditions aren’t met, the input is a letter, which is what you’re looking for! Return the input.

**Validate Input in the Button Event Handler**

1. Inside the event handler function for the Guess button, empty the text of the message element.
2. At the bottom of the event handler, call the function you made that checks the input, and pass it the input value as an argument. Save the result of this function call to a variable and log it out to the console.
3. Use the console to check the input. Enter a character other than a letter into the input. Notice how the message updates on the screen!

**Add a New Global Variable for Player Guesses**

1. Under the word variable, near the top of the code, create another global variable called guessedLetters with an empty array. This array will contain all the letters the player guesses.

**Create a Function to Capture Input**

1. Below the function that checks input, create a new function called makeGuess that accepts a letter as the parameter.
2. JavaScript is case sensitive, so it sees uppercase and lowercase letters as different characters. The easiest way to handle case-sensitivity is to convert all letters to one casing. We recommend converting your letter parameter to uppercase. Once the letter transforms to uppercase, check to see if your guessedLetters array already contains that letter.
3. If the player already guessed the same letter, update the message to inform the player they’ve already guessed that letter and try again. If they haven’t guessed that letter before, add the letter to the guessedLetters array.
4. Log out the guessedLetters array to the console.
5. Return to the event handler for the Guess button. Make sure that the variable mapped to the result of the function validates that the player’s input is returning a letter (as opposed to “undefined”). If it’s returning a letter, pass it as an argument to your makeGuess function.
6. Try a few letter guesses in the browser window. Ensure you’re seeing the guessedLetters array contents updating as you input new letters and click the button.
7. Add and commit your changes with the command line. Push the changes up to GitHub. Copy the link to your repo and submit it below. Part 2 of your project is donzo!

[Challenge Solution(https://github.com/skillcrush/guess-the-word/tree/v02)](https://github.com/skillcrush/guess-the-word/tree/v02)