CREATING THE ANSWER ARRAY

Next we create an empty array called answerArray and fill it with underscores (_) to match the number of letters in the word.

```
  var answerArray = [];
❶ for (var i = 0; i < word.length; i++) {
    answerArray[i] = "_";
  }

  var remainingLetters = word.length;
```

The for loop at ❶ creates a looping variable i that starts at 0 and goes up to (but does not include) word.length. Each time around the loop, we add a new element to answerArray, at answerArray[i]. When the loop finishes, answerArray will be the same length as word. For example, if word is "monkey" (which has six letters), answerArray will be ["_", "_", "_", "_", "_", "_"] (six underscores).

Finally, we create the variable remainingLetters and set it to the length of the secret word. We'll use this variable to keep track of how many letters are left to be guessed. Every time the player guesses a correct letter, this value will be *decremented* (reduced) by 1 for each instance of that letter in the word.

CODING THE GAME LOOP

The skeleton of the game loop looks like this:

```
while (remainingLetters > 0) {
  // Game code goes here
  // Show the player their progress
  // Take input from the player
  // Update answerArray and remainingLetters for every correct guess
}
```

We use a while loop, which will keep looping as long as remainingLetters > 0 remains true. The body of the loop will have to update remainingLetters for every correct guess the player makes. Once the player has guessed all the letters, remainingLetters will be 0 and the loop will end.

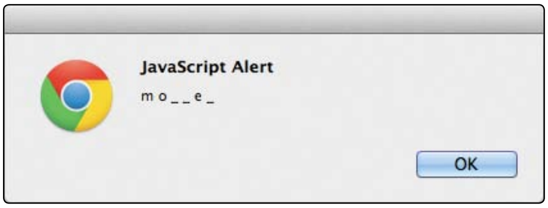
The following sections explain the code that will make up the body of the game loop.

Showing the Player's Progress

The first thing we need to do inside the game loop is to show the player their current progress:

```
alert(answerArray.join(" "));
```

We do that by joining the elements of `answerArray` into a string, using the space character as the separator, and then using `alert` to show that string to the player. For example, let's say the word is *monkey* and the player has guessed *m*, *o*, and *e* so far. The answer array would look like this `["m", "o", "_", "_", "e", "_"]`, and `answerArray.join(" ")` would be `"m o _ _ e _"`. The alert dialog would then look like Figure 7-4.


JavaScript Alert

m o _ _ e _

Figure 7-4: Showing the player's progress using `alert`

Handling the Player's Input

Now we have to get a guess from the player and ensure that it's a single character.

```
❶ var guess = prompt("Guess a letter, or click Cancel to stop playing.");
❷ if (guess === null) {
     break;
❸ } else if (guess.length !== 1) {
     alert("Please enter a single letter.");
   } else {
❹   // Update the game state with the guess
   }
```

At ❶, `prompt` takes a guess from the player and saves it to the variable `guess`. One of four things will happen at this point.

First, if the player clicks the Cancel button, then guess will be null. We check for this condition at ❷ with if (guess === null). If this condition is true, we use break to exit the loop.

NOTE *You can use the* break *keyword in any loop to immediately stop looping, no matter where the program is in the loop or whether the* while *condition is currently* true*.*

The second and third possibilities are that the player enters either nothing or too many letters. If they enter nothing but click OK, guess will be the empty string "". In this case, guess.length will be 0. If they enter anything more than one letter, guess.length will be greater than 1.

At ❸, we use else if (guess.length !== 1) to check for these conditions, ensuring that guess is exactly one letter. If it's not, we display an alert saying, "Please enter a single letter."

The fourth possibility is that the player enters a valid guess of one letter. Then we have to update the game state with their guess using the else statement at ❹, which we'll do in the next section.

UPDATING THE GAME STATE

Once the player has entered a valid guess, we must update the game's answerArray according to the guess. To do that, we add the following code to the else statement:

```
❶ for (var j = 0; j < word.length; j++) {
❷   if (word[j] === guess) {
      answerArray[j] = guess;
❸     remainingLetters--;
    }
  }
```

At ❶, we create a for loop with a new looping variable called j, which runs from 0 up to word.length. (We're using j as the variable in this loop because we already used i in the previous for loop.) We use this loop to step through each letter of word. For example, let's say word is *pancake*. The first time around this loop, when j is 0, word[j] will be "p". The next time, word[j] will be "a", then "n", "c", "a", "k", and finally "e".

At ❷, we use if (word[j] === guess) to check whether the current letter we're looking at matches the player's guess. If it does, we use answerArray[j] = guess to update the answer array with the current guess. For each letter in the word that matches `guess`, we update the answer array at the corresponding point. This works because the looping variable `j` can be used as an index for `answerArray` just as it can be used as an index for `word`, as you can see in Figure 7-5.

```
Index (j)       0    1    2    3    4    5    6
word          " p    a    n    c    a    k    e "
answerArray   ["_", "_", "_", "_", "_", "_", "_"]
```

Figure 7-5: The same index can be used for both `word` and `answerArray`.

For example, imagine we've just started playing the game and we reach the `for` loop at ❶. Let's say `word` is `"pancake"`, `guess` is `"a"`, and `answerArray` currently looks like this:

```
["_", "_", "_", "_", "_", "_", "_"]
```

The first time around the `for` loop at ❶, `j` is 0, so `word[j]` is `"p"`. Our guess is `"a"`, so we skip the `if` statement at ❷ (because `"p" === "a"` is `false`). The second time around, `j` is 1, so `word[j]` is `"a"`. This *is* equal to `guess`, so we enter the `if` part of the statement. The line `answerArray[j] = guess;` sets the element at index 1 (the second element) of `answerArray` to `guess`, so `answerArray` now looks like this:

```
["_", "a", "_", "_", "_", "_", "_"]
```

The next two times around the loop, `word[j]` is `"n"` and then `"c"`, which don't match `guess`. However, when `j` reaches 4, `word[j]` is `"a"` again. We update `answerArray` again, this time setting the element at index 4 (the fifth element) to `guess`. Now `answerArray` looks like this:

```
["_", "a", "_", "_", "a", "_", "_"]
```

The remaining letters don't match `"a"`, so nothing happens the last two times around the loop. At the end of this loop, `answerArray` will be updated with all the occurrences of `guess` in `word`.

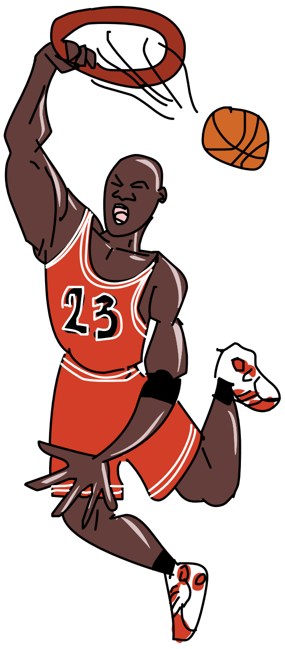
For every correct guess, in addition to updating answerArray, we also need to decrement remainingLetters by 1. We do this at ❸ using remainingLetters--;. Every time guess matches a letter in word, remainingLetters decreases by 1. Once the player has guessed all the letters correctly, remainingLetters will be 0.

ENDING THE GAME

As we've already seen, the main game loop condition is remainingLetters > 0, so as long as there are still letters to guess, the loop will keep looping. Once remainingLetters reaches 0, we leave the loop. We end with the following code:

```
alert(answerArray.join(" "));
alert("Good job! The answer was " + word);
```

The first line uses alert to show the answer array one last time. The second line uses alert again to congratulate the winning player.



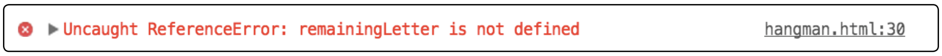
THE GAME CODE

Now we've seen all the code for the game, and we just need to put it together. What follows is the full listing for our Hangman game. I've added comments throughout to make it easier for you to see what's happening at each point. It's quite a bit longer than any of the code we've written so far, but typing it out will help you to become more familiar with writing JavaScript. Create a new HTML file called *hangman.html* and type the following into it:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hangman!</title>
</head>
```

```
<body>
    <h1>Hangman!</h1>

    <script>
    // Create an array of words
    var words = [
      "javascript",
      "monkey",
      "amazing",
      "pancake"
    ];

    // Pick a random word
    var word = words[Math.floor(Math.random() * words.length)];

    // Set up the answer array
    var answerArray = [];
    for (var i = 0; i < word.length; i++) {
      answerArray[i] = "_";
    }

    var remainingLetters = word.length;

    // The game loop
    while (remainingLetters > 0) {
      // Show the player their progress
      alert(answerArray.join(" "));

      // Get a guess from the player
      var guess = prompt("Guess a letter, or click Cancel to stop ↵
playing.");
      if (guess === null) {
        // Exit the game loop
        break;
      } else if (guess.length !== 1) {
        alert("Please enter a single letter.");
      } else {
        // Update the game state with the guess
        for (var j = 0; j < word.length; j++) {
          if (word[j] === guess) {
            answerArray[j] = guess;
            remainingLetters--;
          }
        }
      }
    // The end of the game loop
    }

    // Show the answer and congratulate the player
    alert(answerArray.join(" "));
    alert("Good job! The answer was " + word);
    </script>
</body>
</html>
```

If the game doesn't run, make sure that you typed in everything correctly. If you make a mistake, the JavaScript console can help you find it. For example, if you misspell a variable name, you'll see something like Figure 7-6 with a pointer to where you made your mistake.

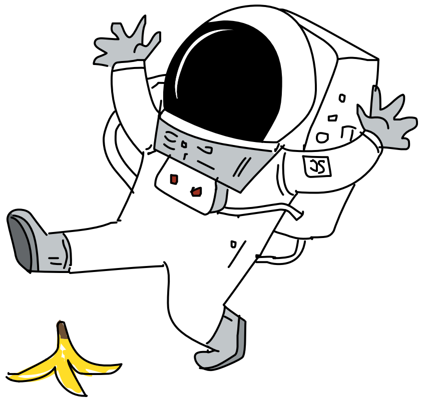


```
▶Uncaught ReferenceError: remainingLetter is not defined        hangman.html:30
```

Figure 7-6: A JavaScript error in the Chrome console

If you click `hangman.html:30`, you'll see the exact line where the error is. In this case, it's showing us that we misspelled `remainingLetters` as `remainingLetter` at the start of the `while` loop.

Try playing the game a few times. Does it work the way you expected it to work? Can you imagine the code you wrote running in the background as you play it?



WHAT YOU LEARNED

In just a few pages, you've created your first JavaScript game! As you can see, loops and conditionals are essential for creating games or any other interactive computer program. Without these control structures, a program just begins and ends.

In Chapter 8, we'll use functions to package up code so you can run it from different parts of your programs.