Learn web development

A first splash into JavaScript

Previous Overview: First steps Next

Now you've learned something about the theory of JavaScript, and what you can do with it, we are going to give you a crash course in the basic features of JavaScript via a completely practical tutorial. Here you'll build up a simple "Guess the number" game, step by step.

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, an understanding of what JavaScript is.
Objective:	To have a first bit of experience at writing some JavaScript, and gain at least a basic understanding of what writing a JavaScript program involves.

You won't be expected to understand all of the code in detail immediately — we just want to introduce you to the high level concepts for now, and give you an idea of how JavaScript (and other programming languages) work. In subsequent articles you'll revisit all these features in a lot more detail!

Note: Many of the code features you'll see in JavaScript are the same as in other programming language — functions, loops, etc. The code syntax looks different, but the concepts are still largely the same.

Thinking like a programmer

One of the hardest things to learn in programming is not the syntax you need to learn, but how to apply it to solve real world problems. You need to start thinking like a programmer — this generally involves looking at descriptions of what your program needs to do and working out what code features

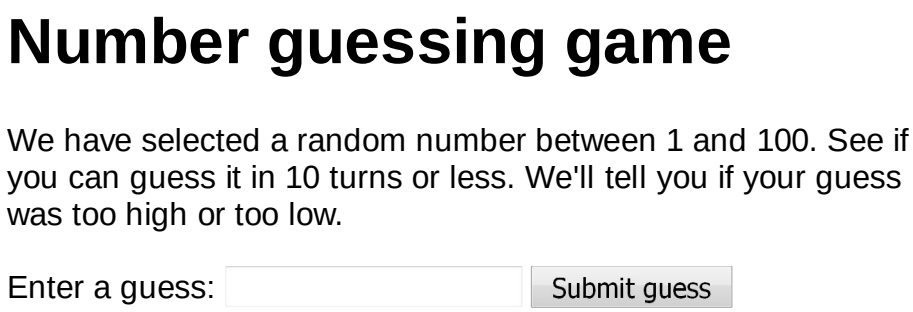
are needed to achieve those things, and how to make them work together.

This requires a mixture of hard work, experience with the programming syntax, and practice — and a bit of creativity. The more you code, the better you'll get at it. We can't promise that you'll develop "programmer brain" in 5 minutes, but we will give you plenty of opportunity to practice thinking like a programmer throughout the course.

With that in mind, let's look at the example we'll be building up in this article, and review the general process of dissecting it into tangible tasks.

Example — Guess the number game

In this article we'll show you how to build up the simple game you can see below:



Have a go at playing it — familiarize yourself with the game before you move on.

Let's imagine your boss has given you the following brief for creating this game:

> "I want you to create a simple guess the number type game. It should choose a random number between 1 and 100, then challenge the player to guess the number in 10 turns. After each turn the player should be told if they are right or wrong, and if they are wrong, whether the guess was too low or too high. It should also tell the player what numbers they previously guessed. The game will end once the player guesses correctly, or once they run out of turns. When the game ends, the player should be given an option to start playing again."

Upon looking at this brief, the first thing we can do is to start breaking it down into simple actionable

tasks, in as much of a programmer mindset as possible:

1. Generate a random number between 1 and 100.
2. Record the turn number the player is on. Start it on 1.
3. Provide the player with a way to guess what the number is.
4. Once a guess has been submitted first record it somewhere so the user can see their previous guesses.
5. Next, check whether it is the correct number.
6. If it is correct:
    1. Display congratulations message.
    2. Stop the player from being able to enter more guesses (this would mess the game up).
    3. Display control allowing the player to restart the game.
7. If it is wrong and the player has turns left:
    1. Tell the player they are wrong.
    2. Allow them to enter another guess.
    3. Increment the turn number by 1.
8. If it is wrong and the player has no turns left:
    1. Tell the player it is game over.
    2. Stop the player from being able to enter more guesses (this would mess the game up).
    3. Display control allowing the player to restart the game.
9. Once the game restarts, make sure the game logic and UI are completely reset, then go back to step 1.

Let's now move forward, looking at how we can turn these steps into code, building up the example, and exploring JavaScript features as we go.

Initial setup

To begin this tutorial, we'd like you to make a local copy of the number-guessing-game-start.html file (see it live here). Open it in both your text editor and your web browser. At the moment you'll see a simple heading, paragraph of instructions and form for entering a guess, but the form won't currently do anything.

The place where we'll be adding all our code is inside the `<script>` element at the bottom of the HTML:

```
<script>

  // Your JavaScript goes here


```

```
    </script>
```

Adding variables to store our data

Let's get started. First of all, add the following lines inside your `<script>` element:

```
var randomNumber = Math.floor(Math.random() * 100) + 1;

var guesses = document.querySelector('.guesses');
var lastResult = document.querySelector('.lastResult');
var lowOrHi = document.querySelector('.lowOrHi');

var guessSubmit = document.querySelector('.guessSubmit');
var guessField = document.querySelector('.guessField');

var guessCount = 1;
var resetButton;
```

This section of the code sets up the variables we need to store the data our program will use. Variables are basically containers for values (such as numbers, or strings of text). You create a variable with the keyword `var` followed by a name for your variable. You can then assign a value to your variable with an equals sign (=) followed by the value you want to give it.

In our example:

- The first variable — `randomNumber` — is assigned a random number between 1 and 100, calculated using a mathematical algorithm.
- The next three variables are each made to store a reference to the results paragraphs in our HTML, and are used to insert values into the paragraphs later on in the code:

```
<p class="guesses"></p>
<p class="lastResult"></p>
<p class="lowOrHi"></p>
```

- The next two variables store references to the form text input and submit button and are used to control submitting the guess later on.

```
<label for="guessField">Enter a guess: </label><input type="text" id="gu
<input type="submit" value="Submit guess" class="guessSubmit">
```

- Our final two variables store a guess count of 1 (used to keep track of how many guesses the player has had), and a reference to a reset button that doesn't exist yet (but will later).

Note: You'll learn a lot more about variables later on in the course, starting with the next article.

Functions

Next, add the following below your previous JavaScript:

```
function checkGuess() {
  alert('I am a placeholder');
}
```

Functions are reusable blocks of code that you can write once and run again and again, saving the need to keep repeating code all the time. This is really useful. There are a number of ways to define functions, but for now we'll concentrate on one simple type. Here we have defined a function by using the keyword `function`, followed by a name, with parentheses put after it. After that we put two curly braces (`{ }`). Inside the curly braces goes all the code that we want to run whenever we call the function.

The code is run by typing the name of the function followed by the parentheses.

Try saving your code now and refreshing it in the browser.

Go into the developer tools JavaScript console, and enter the following line:

```
checkGuess();
```

You should see an alert come up that says "I am a placeholder"; we have defined a function in our code that creates an alert whenever we call it.

Note: You'll learn a lot more about functions later in the course.

Operators

JavaScript operators allow us to perform tests, do maths, join strings together, and other such things.

Let's save our code and refresh the page shown in our browser. Open the developer tools JavaScript console if you haven't got it open already so you can try typing in the examples shown below — type in each one from the "Example" columns exactly as shown, pressing Return/Enter after each one, and see what results they return. If you don't have easy access to the browser developer tools, you can always use the simple built in console seen below:

```
>
```

First let's look at arithmetic operators, for example:

Operator	Name	Example
+	Addition	`6 + 9`
-	Subtraction	`20 - 15`
*	Multiplication	`3 * 7`
/	Division	`10 / 5`

You can also use the + operator to join text strings together (called concatenation in programming). Try entering the following lines:

```
1 var name = 'Bingo';
2 name;
3 var hello = ' says hello!';
4 hello;
5 var greeting = name + hello;
6 greeting;
```

There are also some shortcut operators available, called augmented assignment operators. For example, if you want to simply add a new text string to an existing one and return the result, you could do this:

```
1 name += ' says hello!';
```

This is equivalent to

```
name = name + ' says hello!';
```

When we are running true/false tests (for example inside conditionals — see below, we use comparison operators, for example:

Operator	Name	Example
`===`	Strict equality (is it exactly the same?)	`5 === 2 + 4`
`!==`	Non-equality (is it not the same?)	`'Chris' !== 'Ch' + 'ris'`
`<`	Less than	`10 < 6`
`>`	Greater than	`10 > 20`

Conditionals

Returning to our `checkGuess()` function, I think it's safe to say that we don't want it to just spit out a placeholder message. We want it to check whether a player's guess is correct or not, and respond appropriately.

At this point, replace your current `checkGuess()` function with this version instead:

```
function checkGuess() {
  var userGuess = Number(guessField.value);
  if (guessCount === 1) {
    guesses.textContent = 'Previous guesses: ';
  }
  guesses.textContent += userGuess + ' ';

  if (userGuess === randomNumber) {
    lastResult.textContent = 'Congratulations! You got it right!';
    lastResult.style.backgroundColor = 'green';
    lowOrHi.textContent = '';
    setGameOver();
  } else if (guessCount === 10) {
    lastResult.textContent = '!!!GAME OVER!!!';
    setGameOver();
  } else {
    lastResult.textContent = 'Wrong!';
    lastResult.style.backgroundColor = 'red';

```

```
    if(userGuess < randomNumber) {
      lowOrHi.textContent = 'Last guess was too low!';
    } else if(userGuess > randomNumber) {
      lowOrHi.textContent = 'Last guess was too high!';
    }
  }

  guessCount++;
  guessField.value = '';
  guessField.focus();
}
```

This is a lot of code — phew! Let's go through each section and explain what it does.

- The first line (line 2 above) declares a variable called `userGuess` and sets its value to the current value entered inside the text field. We also run this value through the built-in `Number()` method, just to make sure the value is definitely a number.
- Next, we encounter our first conditional code block (lines 3–5 above). A conditional code block allows you to run code selectively, depending on whether a certain condition is true or not. It looks a bit like a function, but it isn't. The simplest form of conditional block starts with the keyword `if`, then some parentheses, then some curly braces. Inside the parentheses we include a test. If the test returns `true`, we run the code inside the curly braces. If not, we don't, and move on to the next bit of code. In this case the test is testing whether the `guessCount` variable is equal to `1` (i.e. whether this is the player's first go or not):

  ```
  1 | guessCount === 1
  ```

  If it is, we make the guesses paragraph's text content equal to "Previous guesses: ". If not, we don't.
- Line 6 appends the current `userGuess` value onto the end of the `guesses` paragraph, plus a blank space so there will be a space between each guess shown.
- The next block (lines 8–24 above) does a few checks:
  - The first `if(){ }` checks whether the user's guess is equal to the `randomNumber` set at the top of our JavaScript. If it is, the player has guessed correctly and the game is won, so we show the player a congratulations message with a nice green color, clear the contents of the Low/High guess information box, and run a function called `setGameOver()`, which we'll discuss later.
  - Now we've chained another test onto the end of the last one using an `else if(){ }` structure. This one checks whether this turn is the user's last turn. If it is, the program does the same thing as in the previous block, except with a game over message instead of a congratulations message.
  - The final block chained onto the end of this code (the `else { }`) contains code that is only

run if neither of the other two tests returns true (i.e. the player didn't guess right, but they have more guesses left). In this case we tell them they are wrong, then we perform another conditional test to check whether the guess was higher or lower than the answer, displaying a further message as appropriate to tell them higher or lower.

- The last three lines in the function (line 26–28 above) get us ready for the next guess to be submitted. We add 1 to the `guessCount` variable so the player uses up their turn (++ is an incrementation operation — increment by 1), and empty the value out of the form text field and focus it again, ready for the next guess to be entered.

Events

At this point we have a nicely implemented `checkGuess()` function, but it won't do anything because we haven't called it yet. Ideally we want to call it when the "Submit guess" button is pressed, and to do this we need to use an event. Events are actions that happen in the browser, like a button being clicked, or a page loading, or a video playing, in response to which we can run blocks of code. The constructs that listen out for the event happening are called **event listeners**, and the blocks of code run in response to the event firing are called **event handlers**.

Add the following line below the closing curly brace of your `checkGuess()` function:

```
1 guessSubmit.addEventListener('click', checkGuess);
```

Here we are adding an event listener to the `guessSubmit` button. This is a method that takes two input values (called arguments) — the type of event we are listening out for (in this case `click`) as a string, and the code we want to run when the event occurs (in this case the `checkGuess()` function — note that we don't need to specify the parentheses when writing it inside `addEventListener()`).

Try saving and refreshing your code now, and your example should now work, to a point. The only problem now is that if you guess the correct answer or run out of guesses, the game will break because we've not yet defined the `setGameOver()` function that is supposed to be run once the game is over. Let's add our missing code now and complete the example functionality.

Finishing the game functionality

Let's add that `setGameOver()` function to the bottom of our code and then walk through it. Add this now, below the rest of your JavaScript:

```
1 function setGameOver() {
2   guessField.disabled = true;
3   guessSubmit.disabled = true;
4   resetButton = document.createElement('button');
5   resetButton.textContent = 'Start new game';
6
```

```
    document.body.appendChild(resetButton);
    resetButton.addEventListener('click', resetGame);
  }
```

- The first two lines disable the form text input and button by setting their disabled properties to `true`. This is necessary, because if we didn't, the user could submit more guesses after the game is over, which would mess things up.
- The next three lines generate a new `<button>` element, set its text label to "Start new game", and add it to the bottom of our existing HTML.
- The final line sets an event listener on our new button so that when it is clicked, a function called `resetGame()` is run.

Now we need to define this function too! Add the following code, again to the bottom of your JavaScript:

```
function resetGame() {
  guessCount = 1;

  var resetParas = document.querySelectorAll('.resultParas p');
  for (var i = 0 ; i < resetParas.length ; i++) {
    resetParas[i].textContent = '';
  }

  resetButton.parentNode.removeChild(resetButton);

  guessField.disabled = false;
  guessSubmit.disabled = false;
  guessField.value = '';
  guessField.focus();

  lastResult.style.backgroundColor = 'white';

  randomNumber = Math.floor(Math.random() * 100) + 1;
}
```

This rather long block of code completely resets everything to how it was at the start of the game, so the player can have another go. It:

- Puts the `guessCount` back down to 1.
- Clears all the information paragraphs.
- Removes the reset button from our code.

- Enables the form elements, and empties and focuses the text field, ready for a new guess to be entered.
- Removes the background color from the `lastResult` paragraph.
- Generates a new random number so that you are not just guessing the same number again!

At this point you should have a fully working (simple) game — congratulations!

All we have left to do now in this article is talk about a few other important code features that you've already seen, although you may have not realized this.

Loops

One part of the above code that we need to take a more detailed look at is the for loop. Loops are a very important concept in programming, which allow you to keep running a piece of code over and over again, until a certain condition is met.

To start with, go to your browser developer tools JavaScript console again, and enter the following:

```
for (var i = 1 ; i < 21 ; i++) { console.log(i) }
```

What happened? The numbers 1 to 20 were printed out in your console. This is because of the loop. A `for` loop takes three input values (arguments):

1. **A starting value**: In this case we are starting a count at 1, but this could be any number you like. You can replace `i` with any name you like too, but `i` is used as a convention because it's short and easy to remember.
2. **An exit condition**: Here we have specified `i < 21` — the loop will keep going until `i` is no longer less than 21. When `i` reaches 21, the loop will no longer run.
3. **An incrementor**: We have specified `i++`, which means "add 1 to i". The loop will run once for every value of `i`, until `i` reaches a value of 21 (as discussed above). In this case, we are simply printing the value of `i` out to the console on every iteration using `console.log()`.

Now let's look at the loop in our number guessing game — the following can be found inside the `resetGame()` function:

```
var resetParas = document.querySelectorAll('.resultParas p');
for (var i = 0 ; i < resetParas.length ; i++) {
  resetParas[i].textContent = '';
}
```

This code creates a variable containing a list of all the paragraphs inside `<div class="resultParas">` using the `querySelectorAll()` method, then it loops through each one, removing the text content of each.

A small discussion on objects

Let's add one more final improvement before we get to this discussion. Add the following line just below the `var resetButton;` line near the top of your JavaScript, then save your file:

```
guessField.focus();
```

This line uses the `focus()` method to automatically put the text cursor into the `<input>` text field as soon as the page loads, meaning that the user can start typing their first guess right away, and doesn't have to click the form field first. It's only a small addition, but it improves usability — giving the user a good visual clue as to what they've got to do to play the game.

Let's analyze what's going on here in a bit more detail. In JavaScript, everything is an object. An object is a collection of related functionality stored in a single grouping. You can create your own objects, but that is quite advanced and we won't be covering it until much later in the course. For now, we'll just briefly discuss the built-in objects that your browser contains, which allow you to do lots of useful things.

In this particular case, we first created a `guessField` variable that stores a reference to the text input form field in our HTML — the following line can be found amongst our variable declarations near the top:

```
var guessField = document.querySelector('.guessField');
```

To get this reference, we used the `querySelector()` method of the `document` object. `querySelector()` takes one piece of information — a CSS selector that selects the element you want a reference to.

Because `guessField` now contains a reference to an `<input>` element, it will now have access to a number of properties (basically variables stored inside objects, some of which can't have their values changed) and methods (basically functions stored inside objects). One method available to input elements is `focus()`, so we can now use this line to focus the text input:

```
guessField.focus();
```

Variables that don't contain references to form elements won't have `focus()` available to them. For

example, the `guesses` variable contains a reference to a `<p>` element, and `guessCount` contains a number.

Playing with browser objects

Let's play with some browser objects a bit.

1. First of all open your program up in a browser.
2. Next, open your browser developer tools, and make sure the JavaScript console tab is open.
3. Type in `guessField` and the console will show you that the variable contains an `<input>` element. You'll also notice that the console autocompletes object names that exist inside the execution environment, including your variables!
4. Now type in the following:

   ```
   1 guessField.value = 'Hello';
   ```

   The `value` property represents the current value entered into the text field. You'll see that by entering this command, we've changed what that is!
5. Now try typing in `guesses` and pressing return. The console will show you that the variable contains a `<p>` element.
6. Now try entering the following line:

   ```
   1 guesses.value
   ```

   The browser will return `undefined`, because `value` doesn't exist on paragraphs.
7. To change the text inside a paragraph, you need the `textContent` property instead. Try this:

   ```
   1 guesses.textContent = 'Where is my paragraph?';
   ```

8. Now for some fun stuff. Try entering the below lines, one by one:

   ```
   1 guesses.style.backgroundColor = 'yellow';
   2 guesses.style.fontSize = '200%';
   3 guesses.style.padding = '10px';
   4 guesses.style.boxShadow = '3px 3px 6px black';
   ```

   Every element on a page has a `style` property, which itself contains an object whose properties contain all the inline CSS styles applied to that element. This allows us to dynamically set new CSS styles on elements using JavaScript.

Finished for now...

So that's it for building the example — you got to the end, well done! Try your final code out, or play with our finished version here. If you can't get the example to work, check it against the source code.

Previous Overview: First steps Next