A first splash into JavaScript

Now you've learned something about the theory of JavaScript and what you can do with it, we are going to give you an idea of what the process of creating a simple JavaScript program is like, by guiding you through a 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.

We want to set really clear expectations here: You won't be expected to learn JavaScript by the end of this article, or even understand all the code we are asking you to write. Instead, we want to give you an idea of how JavaScript's features work together, and what writing JavaScript feels like. In subsequent articles you'll revisit all the features shown here in a lot more detail, so don't worry if you don't understand it all immediately!

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

Thinking like a programmer

mdn web docs

this generally involves looking at descriptions of what your program needs to do, working out

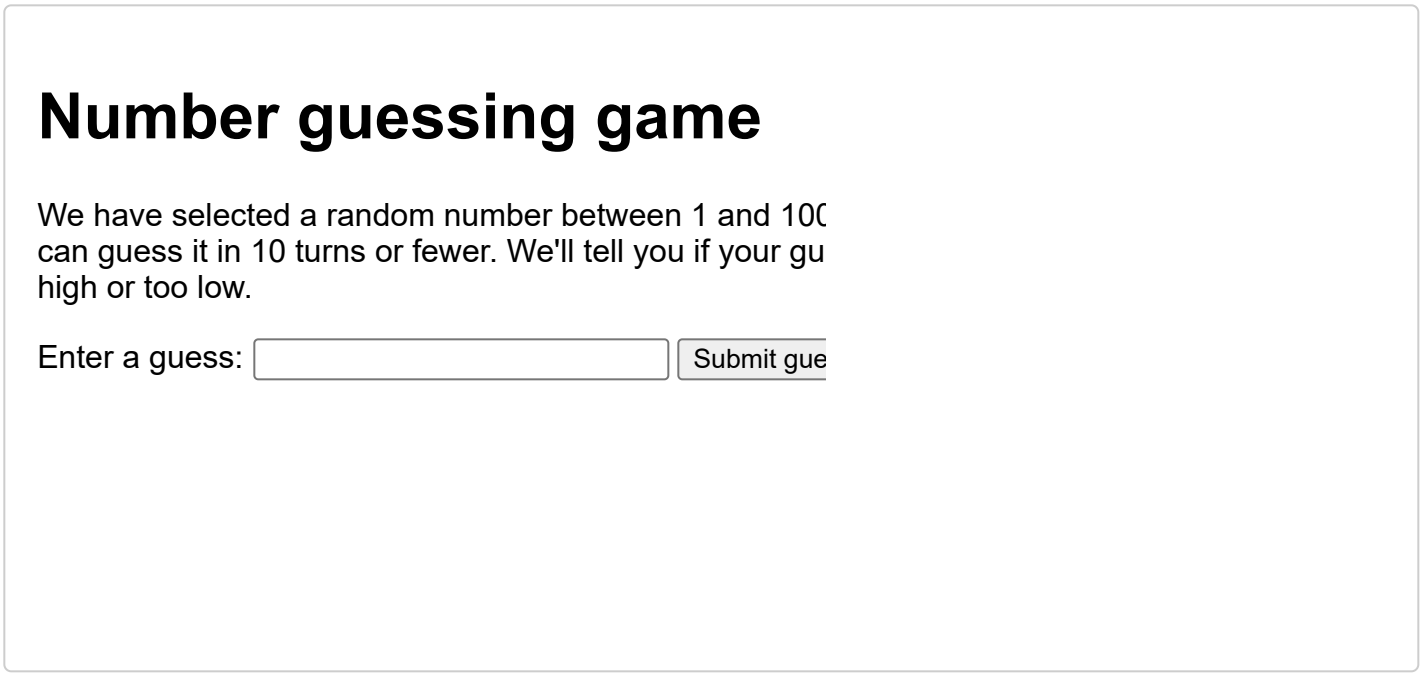
what code features are needed to achieve those things, and how to make them work together.

This requires a mixture of hard work, experience with the programming syntax, and practice — plus 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 five minutes, but we will give you plenty of opportunities 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:
    i. Display congratulations message.
    ii. Stop the player from being able to enter more guesses (this would mess the game up).
    iii. Display control allowing the player to restart the game.
7. If it is wrong and the player has turns left:
    i. Tell the player they are wrong and whether their guess was too high or too low.
    ii. Allow them to enter another guess.
    iii. Increment the turn number by 1.
8. If it is wrong and the player has no turns left:
    i. Tell the player it is game over.
    ii. Stop the player from being able to enter more guesses (this would mess the game up).
    iii. 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:

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

const guesses = document.querySelector(".guesses");
const lastResult = document.querySelector(".lastResult");
const lowOrHi = document.querySelector(".lowOrHi");

const guessSubmit = document.querySelector(".guessSubmit");
const guessField = document.querySelector(".guessField");

let guessCount = 1;
let resetButton;
```

This section of the code sets up the variables and constants we need to store the data our program will use.

Variables are basically names for values (such as numbers, or strings of text). You create a variable with the keyword `let` followed by a name for your variable.

Constants are also used to name values, but unlike variables, you can't change the value once set. In this case, we are using constants to store references to parts of our user interface. The text inside some of these elements might change, but each constant always references the same HTML element that it was initialized with. You create a constant with the keyword `const` followed by a name for the constant.

You can assign a value to your variable or constant 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 first three constants 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 (note how they are inside a `<div>` element, which is itself used to select all three later on for resetting, when we restart the game):

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

- The next two constants 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="number" id="guessField" class="guessField" />
```

```
<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 and constants 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.

When we want to run the code, we type the name of the function followed by the parentheses.

Let's try that now. Save your code and refresh the page in your browser. Then go into the developer tools JavaScript console, and enter the following line:

```
checkGuess();
```

After pressing `Return`/`Enter`, 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 math, join strings together, and other such things.

If you haven't already done so, save your code, refresh the page in your browser, and open the developer tools JavaScript console. Then we 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.

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 (in programming, this is called *concatenation*). Try entering the following lines, one at a time:

```
const name = "Bingo";
name;
const hello = " says hello!";
hello;
const greeting = name + hello;
greeting;
```

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

```
let name1 = "Bingo";
name1 += " says hello!";
```

This is equivalent to

```
let name2 = "Bingo";
name2 = name2 + " 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 // false`
		`'Chris' === 'Bob' // false`
		`5 === 2 + 3 // true`
		`2 === '2' // false; number versus string`

`!==`	Non-equality (is it not the same?)	`5 !== 2 + 4 // true`
		`'Chris' !== 'Bob' // true`
		`5 !== 2 + 3 // false`
		`2 !== '2' // true; number versus string`

`<`	Less than	`6 < 10 // true` `20 < 10 // false`
`>`	Greater than	`6 > 10 // false` `20 > 10 // true`

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() {
  const 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!!!";
    lowOrHi.textContent = "";
    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 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()` constructor, just to make sure the value is definitely a number. Since we're not changing this variable, we'll declare it using `const`.
- Next, we encounter our first conditional code block. 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):

  ```
  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 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 (lines 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 things that happen in the browser — a button being clicked, a page loading, a video playing, etc. — in response to which we can run blocks of code. **Event listeners** observe specific events and call **event handlers**, which are blocks of code that run in response to an event firing.

Add the following line below your `checkGuess()` function:

```
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 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:

```
function setGameOver() {
  guessField.disabled = true;
  guessSubmit.disabled = true;
  resetButton = document.createElement("button");
  resetButton.textContent = "Start new game";
  document.body.append(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;

  const resetParas = document.querySelectorAll(".resultParas p");
  for (const resetPara of resetParas) {
    resetPara.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.
- Empties all the text out of the information paragraphs. We select all paragraphs inside `<div class="resultParas"></div>`, then loop through each one, setting their `textContent` to `''` (an empty string).
- 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 to talk about a few other important code features that you've already seen, although you may have not realized it.

Loops

One part of the above code that we need to take a more detailed look at is the [for...of](#) 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:

```
const fruits = ["apples", "bananas", "cherries"];
for (const fruit of fruits) {
```

```
  console.log(fruit);
}
```

What happened? The strings `'apples', 'bananas', 'cherries'` were printed out in your console.

This is because of the loop. The line `const fruits = ['apples', 'bananas', 'cherries'];` creates an array. We will work through a complete Arrays guide later in this module, but for now: an array is a collection of items (in this case strings).

A `for...of` loop gives you a way to get each item in the array and run some JavaScript on it. The line `for (const fruit of fruits)` says:

1. Get the first item in `fruits`.
2. Set the `fruit` variable to that item, then run the code between the `{}` brackets.
3. Get the next item in `fruits`, and repeat 2, until you reach the end of `fruits`.

In this case, the code inside the brackets is writing out `fruit` to the console.

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

```
const resetParas = document.querySelectorAll(".resultParas p");
for (const resetPara of resetParas) {
  resetPara.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.

Note that even though `resetPara` is a constant, we can change its internal properties like `textContent`.

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 `let 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, without having 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, most of the items you will manipulate in your code are objects. 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` constant that stores a reference to the text input form field in our HTML — the following line can be found amongst our declarations near the top of the code:

```
const 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 now has 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` constant contains a reference to a `<p>` element, and the `guessCount` variable contains a number.

Playing with browser objects

Let's play with some browser objects a bit.

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

   ```
   guessField.value = 2;
   ```

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

   ```
   guesses.value;
   ```

   The browser returns `undefined`, because paragraphs don't have the `value` property.
7. To change the text inside a paragraph, you need the `textContent` property instead. Try this:

   ```
   guesses.textContent = "Where is my paragraph?";
   ```

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

```
guesses.style.backgroundColor = "yellow";
guesses.style.fontSize = "200%";
guesses.style.padding = "10px";
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.

This page was last modified on May 9, 2023 by MDN contributors.