Anatomy of Javascript

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the parts of JavaScript by discussing:

1. REVIEW: HOW JAVASCRIPT IS USED IN WEB PAGES
2. JAVASCRIPT SYNTAX PREVIEW
3. VARIABLES AND VALUES
4. CONDITIONALS AND LOOPS

1. REVIEW: HOW JAVASCRIPT IS USED IN WEB PAGES

In Unit 1, we showed an example of using JavaScript to make a paragraph element interactive and clickable. You also learned how browsers take in HTML, CSS, and JavaScript source code and execute the code, in order, to create a web page. We also covered how developers use JavaScript APIs like browser APIs and third-party APIs to build complex applications and went a bit into how browsers execute JavaScript.

Since JavaScript is a programming language, it's significantly more complex compared to HTML and CSS. There are entire, multi-week-long courses devoted to helping engineers learn JavaScript basics. Don't worry — we don't assume that you want to embark on that journey...*yet*. Instead, we hope that the content in this section will pique your interest and inspire you to continue learning JavaScript.

2. JAVASCRIPT SYNTAX PREVIEW

JavaScript belongs to a family of programming languages known as **C-like languages**— languages inspired by the C programming language — so you'll find that it shares features with other C-like languages like Python and Ruby. We'll give you a high-level overview of some of these features since you'll likely encounter them again, even if you decide to learn a different programming language.

JavaScript allows you to create and manipulate data like text and numbers, save their values in variables, control the flow of logic in a program with conditionals and loops, organize complex data in data structures, and create reusable portions of code. You might not understand what this means yet, but don't worry we'll step through it together. Here's an example JavaScript program:

alert(“Hello, world!”);
In the line above, **alert** is the name of a function that displays alerts to the user. You can use **alert** by adding parentheses after its name. Then, inside the parentheses, we specify what the alert should say — in this case, the alert will contain a message reading **Hello, world!**. The semicolon at the end of the line tells JavaScript that this is the end of the statement.

---

3. VARIABLES AND VALUES

“Hello, world!” is a **value**. Other values include numbers like **100**, values that represent true and false, and even values that are groups of code. Values can be assigned to variables. Variables are created with the keywords **const** or **let**.

```
const greeting = “Hello, world!”;
let luckyNum = 100;
```

Variables allow you to reuse values. For example, since we’ve saved **“Hello, world!”** to the variable **greeting**, we can use the variable with the **alert** function:

```
alert(greeting);
```

Values that are groups of code are called **functions**. The **alert** function is a group of code that is built into the browser and can be used to display alerts to the user. You can also define your own function and save it to a variable so that you can repurpose it later. For example:

```
const sendGreeting = function () {
  alert(“Hello, world!”);
};
```

This will allow you to send a Hello, world! alert again by writing the function’s name followed by a pair of parentheses:

```
sendGreeting();
```

Add code to the *Editable Code* box below to display an alert that says **1 + 2 is 3**. You’ll do this by reusing a function stored in the variable **addNums**. You can use the *Click to Run Code* button to execute your code. If you want to start over, click *Reset*. Click *Show Solution* to show the solution.

TRY IT

Follow these steps in the box below:

1. Remove the line that says // **Replace this** and, in its place, write the name of the function, **addNums**.

2. Then add an opening parenthesis **(**.

3. List the two numbers you want to add, separated by a comma. Since you want to add **1** and **2**, you should write **1, 2**.

4. Add the closing parenthesis **)** and end the statement with a semicolon **;**.

5. Press *Click to Run Code* to verify that everything works! You'll see an alert dialog show in your browser if the code is correct.}}

4. CONDITIONALS AND LOOPS

There's also syntax to control the flow of logic throughout a program based on a condition being true or false. For example, the if statement will execute code **if** a condition is true.

```
(const myNum = 100;

if (myNum > 500) {

  alert("That's a large number.");
}
```

The code above will send an alert if **myNum** is greater than **500**. Since **myNum** is **100** and **100** is not greater than **500**, **myNum > 500** is false and the user will not receive an alert. We can combine **if** with an **else** statement. An else statement tells the program what to do when the if statement is not true.

```
const myNum = 100;

if (myNum > 500) {
  alert("That's a large number.");
} else {
  alert("myNum is less than 500.");
}
```

Now the user will receive a different alert based on if **myNum** is greater than **500** or not. Since **myNum > 500** is false, the user's alert will say **myNum is less than 500**.

It seems a bit too boring to send an alert that says **myNum is less than 500** when myNum is smaller than 500. It'd be more exciting if we sent an alert when myNum is greater than 500 or when myNum is a negative number.

Edit the code in the *Editable Code* so the user gets an alert if myNum is greater than 500 or if myNum is less than 0. If myNum is less than 0, send an alert that says **Whoa, a negative number!**. Notice there are two conditions to check, myNum>500 and myNum<0.

You will need an **else if** statement instead of an else statement. **else if** statements must go after **if** statements and before any **else** statements (if they exist). We don't need the **else** statement any more, so you'll replace it with an **else if.**

TRY IT

Follow these steps in the box below:

1. Add a space after **else** and type **if**.

2. Add another space and an opening parenthesis **(**.

3. Write the second condition **myNum < 0** after the opening parenthesis.

4. Close everything up with a closing parenthesis **)**.

5. Remember, *Click to Run Code* will run your code, *Reset* will reset the workspace, and *Show Solution* will show you the solution.

Another way to control a program's flow is to use a **while** statement or a **for** statement to repeat a block of code.

```
let count = 0;
while (count < 3) {
  alert(count);
  count = count + 1;
}
```

The example above uses the **while** statement to send an alert that displays the value of the **count** variable. The **while** statement will evaluate the condition — in this case the condition is **count<3** — and execute the code inside the **while** statement if the condition is true. The code above will send three alerts — an alert that says **0**, an alert that says **1**, and an alert that says **2**.

We can also rewrite the code above so it does the exact same thing but with a **for** statement instead of a **while** statement:

```
for (let count = 0; count < 3; count = count + 1) {
  alert(count);
}
```

The example above will also send three alerts — an alert that says **0**, an alert that says **1**, and an alert that says **2** — and follows the same flow of logic as the code in the example **while** statement.

Here's a step-by-step breakdown of how JavaScript will execute the example code with the **while** statement and the one with the for statement:

1. Create a variable named **count** and set its value to **0**.

2. Evaluate count < 3 since it evaluates to true, execute the code inside the statement:

- Send an alert that says **0**
- Reassign the value of **count** to **1**

3. Evaluate count < 3 since it evaluates to true, execute the code inside the statement:

- Send an alert that says **1**
- Reassign the value of **count** to **2**

4. Evaluate count < 3 since it evaluates to true, execute the code inside the statement:

- Send an alert that says **2**
- Reassign the value of **count** to **3**

5. Evaluate count < 3 since it evaluates to false, terminate the loop.
