Internet and Web Principal

Week 9

JavaScript 1

Content

1. What is JavaScript?
2. Variables and arrays
3. If/ else statements and loops

What is JavaScript?

JavaScript is a computer language that enhances our websites with interactivity and specific behavior. It is a client-side scripting language, which means it operates on the user's computer rather than the server, like other web programming languages like PHP and Ruby do. That implies JavaScript (and how we use it) is dependent on the capabilities and settings of the browser. It may not be available at all, either because the user has disabled it or because the device does not support it, which is something that excellent developers consider and plan for. JavaScript is another name for a dynamic and loosely typed programming language.

What is JavaScript?

JavaScript is a small yet extremely powerful scripting language. JavaScript is most often encountered in browsers, but it has crept into everything from native programs to PDFs to ebooks. JavaScript can even be used to power web servers. JavaScript, as a dynamic programming language, does not require any type of compiler to translate our human-readable code into something the browser can comprehend. The browser reads the code in the same way that we do and interprets it on the fly.

What is JavaScript?

JavaScript is most frequently used to provide interactivity to a page. Whereas HTML markup makes up the "structural" layer of a page and CSS makes up the "presentational" layer, JavaScript makes up the third "behavioral" layer. Scripts may access all of the components, attributes, and text on a web page via the DOM (Document Object Model). We may also develop scripts that respond to user input by changing the page's content, CSS styles, or browser behavior on the fly.

What is JavaScript?

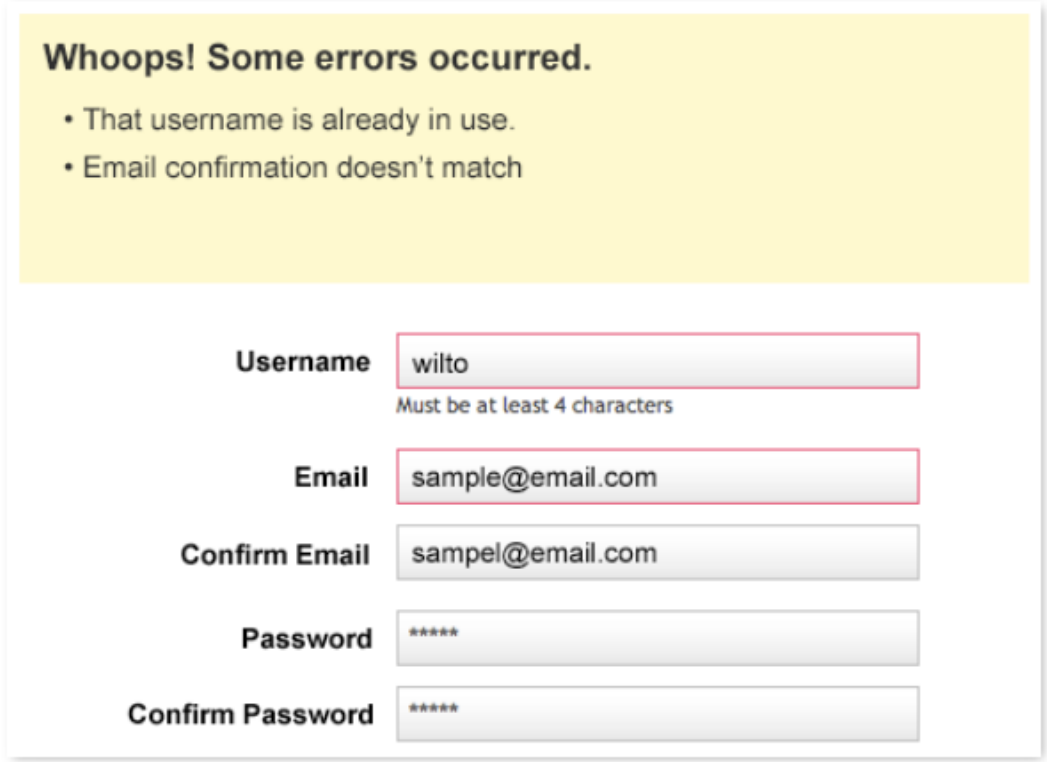

FIGURE 21-1. JavaScript inserts a message, alters styles to make errors apparent, and blocks the form from submitting. It can also detect whether the email entries match, but the username would more likely be detected by a program on the server.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 595

What is JavaScript?

In short, JavaScript allows you to create highly responsive interfaces that improve the user experience and provide dynamic functionality, without waiting for the server to load up a new page. For example, we can use JavaScript to do any of the following:

- Suggest the complete term a user might be entering in a search box as he types. You can see this in action on Google.com (FIGURE 21-2).

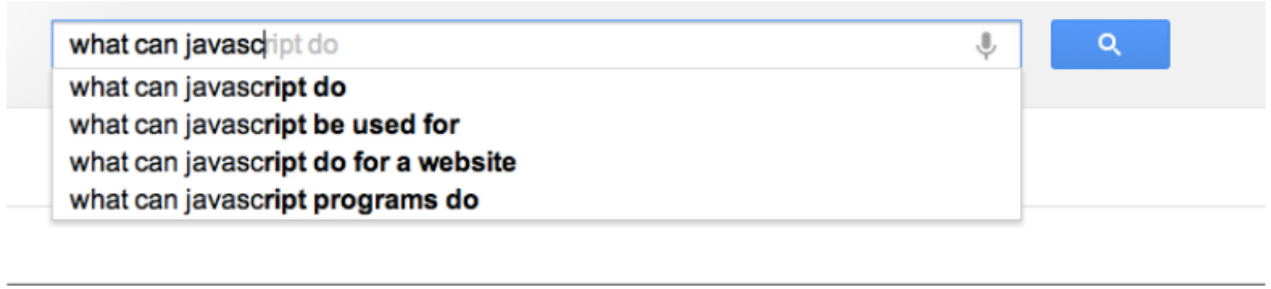

FIGURE 21-2. Google.com uses JavaScript to automatically complete a search term as it is typed in.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 596

What is JavaScript?

- Show and hide content based on a user clicking a link or heading, to create a "collapsible" content area (FIGURE 21-3).

FIGURE 21-3. JavaScript can be used to reveal and hide portions of content.

- Test for browsers' individual features and capabilities. For example, one can test for the presence of "touch events," indicating that the user is interacting with the page through a mobile device's browser, and add more touch-friendly styles and interaction methods.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 586

What is JavaScript?

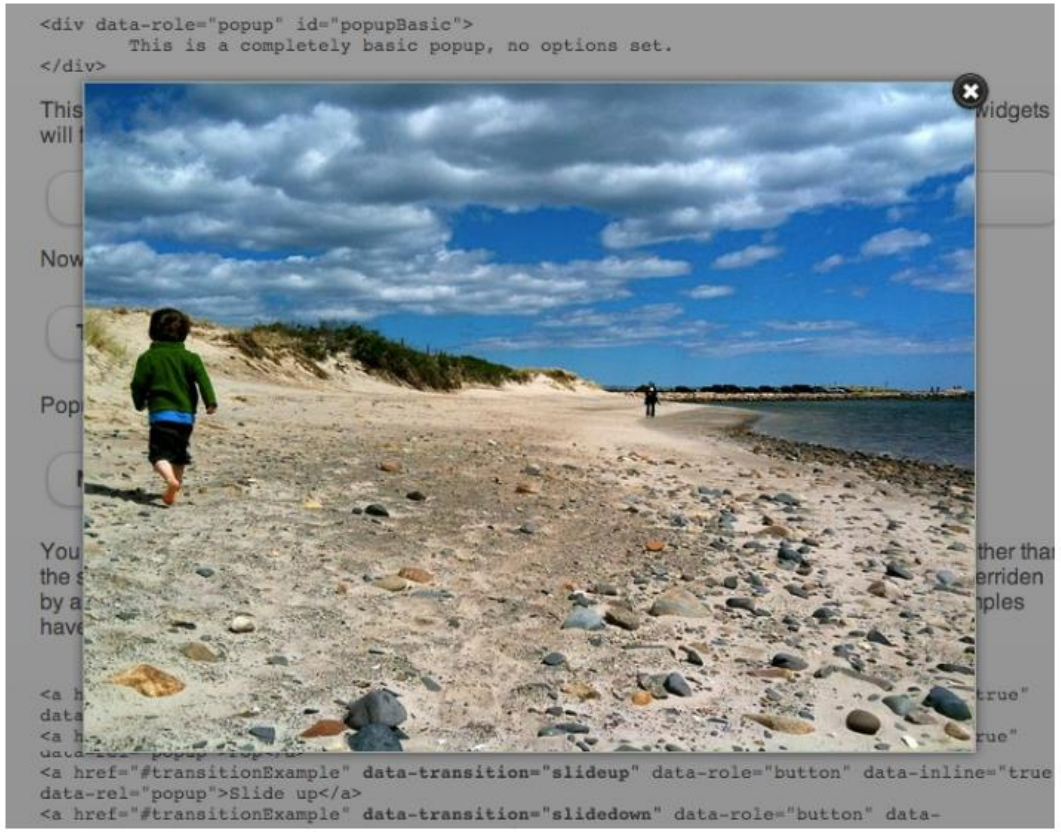

FIGURE 21-4. JavaScript can be used to load images into a lightbox-style gallery.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 597

What is JavaScript?

ADDING JAVASCRIPT TO A PAGE

As with CSS, you can embed a script right in a document or keep it in an external file and link it to the page. Both methods use the **script** element.

Embedded Script

To embed a script on a page, just add the code as the content of a **script** element:

```
<script>
... JavaScript code goes here
</script>
```

External Scripts

The other method uses the **src** attribute to point to a script file (with a *.js* suffix) by its URL. In this case, the **script** element has no content:

```
<script src="my_script.js"></script>
```

The advantage to external scripts is that you can apply the same script to multiple pages (the same benefit external style sheets offer). The downside, of course, is that each external script requires an additional HTTP request of the server, which slows down performance.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 598

What is JavaScript?

Script Placement

The script element can go anywhere in the document, but the most com- mon places for scripts are in the head of the document and at the very end of the body. It is recommended that you don’t sprinkle them throughout the document, because they would be difficult to find and maintain.

For most scripts, the end of the document, just before the </body> tag, is the preferred placement because the browser will be done parsing the document and its DOM structure:

Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, Page 598

What is JavaScript?

```
<!DOCTYPE html> <html lang="en"> <head>
<meta charset="utf-8"> </head>
<body>
...contents of page...
<script src="script.js"></script>
</body> </html>
```

Consequently, that information will be ready and available by the time it gets to the scripts, and they can execute faster. In addition, the script download and execution blocks the rendering of the page, so moving the script to the bottom improves the perceived performance.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 598

Variables and Arrays

The Fundamentals

There are a few basic syntactical rules that run throughout JavaScript.

It is critical to understand that JavaScript is case-sensitive. Variables called MyVariable, myVariable, and MYVariable will be considered as three different objects. Also, whitespace such as tabs and spaces are disregarded unless they are part of a text string and wrapped in quotes. All of the character gaps added to scripts, such as those in this chapter, are for the advantage of humans—they make it simpler to understand the code. They are invisible to JavaScript.

Variables and Arrays

Statements

A script is a collection of statements. A statement is a command that instructs the browser. Here's a simple line that causes the browser to display a "Thank you" alert:

```
alert("Thank you.");
```

The semicolon at the conclusion of the statement notifies JavaScript that the command has ended, much like a period does at the end of a sentence. A line break, according to the JavaScript standard, will also signal the conclusion of a command, however it is best practice to conclude each statement with a semicolon.

Variables and Arrays

Comments

To give reminders and explanations throughout your code, JavaScript allows you to leave comments that will be disregarded when the script is performed. This is especially useful if the code will be modified by another developer in the future.

Case is important in JavaScript.

Variables and Arrays

Variables

A variable functions similarly to an information container. You give it a name and then assign it a value, which can be a number, text string, DOM element, function, or anything else. This allows us to easily refer to that value later by name. The value itself can be changed and reallocated in any way that our scripts' logic requires.

The following statement declares a variable called nilai and assigns it the value 5:

```
var nilai = 5;
```

Variables and Arrays

To begin, we use the var keyword to declare the variable. The single equals symbol (=) signifies that we are putting a value on it. We terminate the line with a semicolon since it is the conclusion of our sentence. Variables can also be declared without the var keyword, which affects which parts of your script have access to the data they contain.

Variables and Arrays

You may use whatever you want as a variable name, but make sure it makes sense to you afterwards. You wouldn't name a variable data; instead, it should describe the information it holds. In our previous, more particular example, productName may be a more helpful name. There are a few rules for naming a variable:

- It must start with a letter or an underscore.
- It may contain letters, digits, and underscores in any combination.
- It may not contain character spaces. As an alternative, use underscores in place of spaces, or close up the space and use camel case instead (for example, my_variable or myVariable).
- It may not contain special characters (e.g., ! . , / \ + * =).

Variables and Arrays

Data types

The values we assign to variables fall under a few distinct data types:

Undefined

The simplest of these data types is likely **undefined**. If we declare a vari- able by giving it a name but no value, that variable contains a value of **undefined**.

```
var foo;
alert(foo); // This will open a dialog containing "undefined".
```

Variables and Arrays

Null

Similar to **undefined**, assigning a variable of **null** (again, case-sensitive) simply says, “Define this variable, but give it no inherent value.”

```
var foo = null;
alert(foo); // This will open a dialog containing "null".
```

Numbers

You can assign variables numeric values.

```
var foo = 5;
alert(foo); // This will open a dialog containing "5".
```

Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, Page 601

Variables and Arrays

Strings

Another type of data that can be saved to a variable is a string, which is basically a line of text. Enclosing characters in a set of single or double quotes indicates that it's a string, as shown here:

```
var foo = "five";
alert( foo ); // This will alert "five"
```

The variable **foo** is now treated exactly the same as the word *five*. This applies to any combination of characters: letters, numbers, spaces, and so on. If the value is wrapped in quotation marks, it will be treated as a string of text. If we were to wrap the number 5 in quotes and assign it to a variable, that variable wouldn't behave as a number; instead, it would behave as a string of text containing the character "5."

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 602

Variables and Arrays

Earlier we saw the plus sign (**+**) used to add numbers. When the plus sign is used with strings, it sticks the strings together (called concatenation) into one long string, as shown in this example.

```
var foo = "bye"
alert(foo + foo); // This will alert "byebye"
```

Notice what the alert returns in the following example when we define the value 5 in quotation marks, treating it as a string instead of a number:

```
var foo = "5";
alert( foo + foo ); // This will alert "55"
```

If we concatenate a string and a number, JavaScript will assume that the number should be treated as a string as well, since the math would be impossible.

```
var foo = "five";
var bar = 5;
alert( foo + bar ); // This will alert "five5"
```

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 602

Variables and Arrays

Booleans

We can also assign a variable a true or false value. This is called a Boolean value, and it is the lynchpin for all manner of advanced logic. Boolean val- ues use the **true** and **false** keywords built into JavaScript, so quotation marks are not necessary.

```
var foo = true; // The variable "foo" is now true
```

Just as with numbers, if we were to wrap the preceding value in quotation marks, we'd be saving the word *true* to our variable instead of the inherent value of **true**

In a sense, everything in JavaScript has either an inherently true or false value. For example, **null**, **undefined**, **0**, and empty strings (**" "**) are all inherently false, while every other value is inherently true.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 602

Variables and Arrays

Arrays

An array is a group of multiple values (called members) that can be assigned to a single variable. The values in an array are said to be indexed, meaning you can refer to them by number according to the order in which they appear in the list. The first member is given the index number 0, the second is 1, and so on, which is why one almost invariably hears us nerds start counting things at zero—because that's how JavaScript counts things, and many other programming languages do the same. We can avoid a lot of future coding headaches by keeping this in mind.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 602

Variables and Arrays

So, let's say our script needs all of the variables we defined earlier. We could define them three times and name them something like **foo1**, **foo2**, and so on, or we can store them in an array, indicated by square brackets (**[ ]**).

```
var foo = [5, "five", "5"];
```

Now anytime you need to access any of those values, you can grab them from the single **foo** array by referencing their index number:

```
alert( foo[0] ); // Alerts "5"
alert( foo[1] ); // Alerts "five"
alert( foo[2] ); // Also alerts "5"
```

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 603

Variables and Arrays

Comparison Operators

Now that we know how to save values to variables and arrays, the next logi- cal step is knowing how to compare those values. There is a set of special characters called comparison operators that evaluate and compare values in different ways:

==	Is equal to
!=	Is not equal to
===	Is identical to (equal to and of the same data type) **!==** Is not identical to
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 603

Variables and Arrays

There’s a reason all of these definitions read as parts of a statement. In comparing values, we’re making an assertion, and the goal is to obtain a result that is either inherently true or inherently false. When we compare two values, JavaScript evaluates the statement and gives us back a Boolean value depending on whether the statement is true or false.

```
alert( 5 == 5 ); // This will alert "true"
alert( 5 != 6 ); // This will alert "true"
alert( 5 < 1 ); // This will alert "false"
```

Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, Page 603

Variables and Arrays

Equal versus identical

The tricky part is understanding the difference between “equal to” (**==**) and “identical to” (**===**). We already learned that all of these values fall under a certain data type. For example, a string of “5” and a number 5 are similar, but they’re not quite the same thing.

Well, that’s exactly what **===** is meant to check.

```
alert( "5" == 5 ); // This will alert "true". They're both "5".

alert( "5" === 5 );

/* This will alert "false". */

alert( "5" !== 5 );

/* This will alert "true", since they're not the same data type. */
```

Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, Page 603

Variables and Arrays

Mathematical operators

The other type of operator is a mathematical operator, which performs mathematical functions on numeric values (and, of course, variables that contain numeric values). We touched briefly on the straightforward mathematical operators for add (**+**), subtract (**-**), multiply (*), and divide (**/**). There are also some useful shortcuts you should be aware of:

+= Adds the value to itself

++ Increases the value of a number (or a variable containing a number value) by 1

-- Decreases the value of a number (or a variable containing a number value) by 1

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 604

If/else statements and loops

if/else statements

if/else statements are how we get JavaScript to ask itself a true/false question. They are more or less the foundation for all the advanced logic that can be written in JavaScript, and they're about as simple as programming gets. In fact, they're almost written in plain English. The structure of a conditional statement is as follows:

```
if( true ) {
// Do something.
}
```

It tells the browser "if this condition is met, then execute the commands listed between the curly brackets (**{ }**)." JavaScript doesn't care about whitespace in our code, remember, so the spaces on either side of the **(true)** are purely for the sake of more readable code.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 604

If/else statements and loops

Here is a simple example using the array we declared earlier:

```
var foo = [5, "five", "5"];
if( foo[1] === "five" ) {
alert("This is the word five, written in plain English.");
}
```

Since we're making a comparison, JavaScript is going to give us a value of either **true** or **false**. The highlighted line of code says "true or false: the value of the **foo** variable with an index of **1** is identical to the word 'five'?"

In this case, the alert would fire because the **foo** variable with an index of **1** (the second in the list, if you'll remember) is identical to "five". It is indeed true, and the alert fires.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 605

If/else statements and loops

That covers “if,” but what about “else”?

what if we want to do one thing if something is true and something *else* if that thing is false? We could write two **if** statements, but that’s a little clunky. Instead, we can just say, “else, do something...else.”

```
var test = "testing";
if( test == "testing" ) {
alert( "You haven't changed anything." ); } else {
alert( "You've changed something!" ); }
```

Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, Page 605

If/else statements and loops

Loops

There are several ways to write a loop, but the **for** method is one of the most popular. The basic structure of a **for** loop is as follows:

for(*initialize the variable*; *test the condition*; *alter the value*;) {

// do something

}

Here's an example of a **for** loop in action:

```
for( var i = 0; i < 2; i++ ) {
alert( i ); // This loop will trigger three alerts, reading "0",
"1", and "2" respectively.
}
```

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 605

If/else statements and loops

- **for()**
- First, we're calling the **for()** statement, which is built into JavaScript. It says, "For every time this is true, do this." Next we need to supply that statement with some information.
- **var i = 0;**
- This creates a new variable, **i**, with its value set to zero. You can tell it's a variable by the single equals sign. More often than not, you'll see coders using the letter "i" (short for "index") as the variable name, but keep in mind that you could use any variable name in its place. It's a common convention, not a rule.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 605

If/else statements and loops

i++

Finally, **i++** is shorthand for “every time this loop runs, add one to the value of **i**” (**++** is one of the mathematical shortcut operators we saw earlier). Without this step, **i** would always equal zero, and the loop would run forever! Fortunately, modern browsers are smart enough not to let this happen. If one of these three pieces is missing, the loop simply won’t run at all.

Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, Page 605

If/else statements and loops

{ *script* **}**

Anything inside those curly brackets is executed once for each time the loop runs, which is three times in this case. That **i** variable is available for use in the code the loop executes as well, as we'll see next.

Let's go back to the "check each item in an array" example. How would we write a loop to do that for us?

```
var items = ["foo", "bar", "baz"]; // First we create an array. for( var i = 0; i < items.length; i++ ) {

alert( items[i] ); // This will alert each item in the array. }
```

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 605

If/else statements and loops

items.length

Instead of using a number to limit the number of times the loop runs, we're using a property built right into JavaScript to determine the "length" of our array, which is the number of items it contains. **.length** is just one of the standard properties and methods of the **Array** object in JavaScript. In our example, there are three items in the array, so it will loop three times.

items[i]

Remember how I mentioned that we can use that **i** variable inside the loop? Well, we can use it to reference each index of the array. Good thing we started counting from zero; if we had set the initial value of **i** to 1, the first item in the array would have been skipped. The result of our **for** loop example is that each item in the array (the text strings **foo**, **bar**, and **baz**) gets returned after each loop and fed to an alert.

Learning Web Design, Jennifer Niederst Robbins, O'Reilly Media, Inc., 2018, Page 605

Thank You

Alfred Tenggono, S.Kom., M.Kom.

alfred.tenggono@jiu.ac

Reference

- Learning Web Design, Jennifer Niederst Robbins, O’Reilly Media, Inc., 2018, ISBN: 978-1-491-96020-2