21
INTRODUCTION TO JAVASCRIPT

OVERVIEW

- What JavaScript is
- Variables and arrays
- if/else statements and loops
- Native and custom functions
- Browser objects
- Event handlers

What Is JavaScript?

- JavaScript is a **client-side scripting language**—it is processed on the user's machine (not the server).
- It is reliant on the browser's capabilities (it may even be unavailable entirely).
- It is a **dynamic programming language**—it does not need to be compiled into an executable program. The browser reads it just as we do.
- It is **loosely typed**—you don't need to define variable types as you do for other programming languages.

JavaScript Tasks

JavaScript adds a **behavioral layer** (interactivity) to a web page. Some examples include:

- Checking form submissions and provide feedback messages and UI changes
- Injecting content into current documents on the fly
- Showing and hiding content based on a user clicking a link or heading
- Completing a term in a search box
- Testing for browser features and capabilities
- Much more!

Adding Scripts to a Page

Embedded script
Include the script in an HTML document with the **`script`** element:

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

External script
Use the **`src`** attribute in the **`script`** element to point to an external, standalone *.js* file:

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

Script Placement

The **script** element can go anywhere in the document, but the most common places are as follows:

In the **head** of the document

For when you want the script to do something before the body completely loads (ex: Modernizr):

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

Just before the **</body>** tag

Preferred when the browser needs to parse the document and its DOM structure before running the script:

```
...
<body>
  <!--contents of page-->
<script src="script.js"></script>
</body>
</html>
```

JavaScript Syntax Basics

- JavaScript is **case-sensitive**.
- **Whitespace is ignored** (unless it is enclosed in quotes in a text string).
- A script is made up of a series of **statements**, commands that tell the browser what to do.
- **Single-line comments** in JavaScript appear after two `//` characters:

```
// This is a single-line comment
```

- **Multiple-line comments** go between `/*` and `*/` characters.

Building Blocks of Scripts

- Variables
- Comparison operators
- if/else statements
- Loops
- Functions
- Scope

Variables

- A **variable** is made up of a **name** and a **value**.
- You create a variable so that you can refer to the value by name later in the script.
- The value can be a number, text string, element in the DOM, or function, to name a few examples.
- Variables are defined using the **`var`** keyword:

```
var foo = 5;
```

The variable is named `foo`. The equals sign (=) indicates we are **assigning** it the numeric value of 5.

Variables (cont’d)

Rules for naming a variable:

- It must start with a letter or underscore.
- It may not contain character spaces. Use underscores or CamelCase instead.
- It may not contain special characters (`! . , / \ + * =`).
- It should describe the information it contains.

Value Data Types

Values assigned to variables fall under a few **data types**:

Undefined

The variable is declared by giving it a name, but no value:

```
var foo;
alert(foo); // Will open a dialog containing "undefined"
```

`null`

Assigns the variable no inherent value:

```
var foo = null;
alert(foo); // Will open a dialog containing "null"
```

Numbers

When you assign a number (e.g., 5), JavaScript treats it as a number (you don’t need to tell it it's a number):

```
var foo = 5;
alert(foo + foo); // This will alert "10"
```

Value Data Types (cont'd)

Strings

If the value is wrapped in single or double quotes, it is treated as a string of text:

```
var foo = "five";
alert(foo); // Will alert "five"
alert(foo + foo); // Will alert "fivefive"
```

Booleans

Assigns a true or false value, used for scripting logic:

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

Arrays

A group of multiple values (called *members*) assigned to a single variable. Values in arrays are *indexed* (assigned a number starting with 0). You can refer to array values by their index numbers:

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

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

Comparison Operators

Comparison operators are special characters in JavaScript syntax that evaluate and compare values:

`==`	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

Comparison Operators (cont’d)

Example
When we compare two values, JavaScript evaluates the statement and gives back a Boolean (true/false) value:

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

NOTE: Equal to (**==**) is not the same as identical to (**===**). Identical values must share a data type:

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

alert( "5" === 5 ); // This will alert "false". They're both
"5", but they're not the same data type.

alert( "5" !== 5 ); // This will alert "true", since they're
not the same data type.
```

Mathematical Operators

Mathematical operators perform mathematical functions on numeric values:

`+`	Add
`-`	Subtract
`*`	Multiply
`/`	Divide
`+=`	Adds the value to itself
`++`	Increases the value of a number (or number in a variable) by 1
`--`	Decreases the value of a number (or number in a variable) by 1

if/else Statements

An **if/else statement** tests for conditions by asking a true/false question.

If the condition in parentheses is met, then execute the commands between the curly brackets (**{}**):

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

Example:

```
if( 1 != 2 ) {
  alert("These values are not equal.");
   // It is true that 1 is never equal to 2, so we should see
this alert.
}
```

if/else Statements (cont’d)

If you want to do one thing if the test is true and something else if it is false, include an **else statement** after the if statement:

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

Changing the value of the test variable to anything but the word “testing” will trigger the alert “You've changed something!”

Loops

Loops allow you to do something to every variable in an array without writing a statement for every one.

One way to write a loop is with a **for statement**:

```
for(initialize variable; test condition; alter the value;) {
   // do something
}
```

Loops (cont'd)

Example: This loop triggers **3 alerts**, reading "0", "1", and "2":

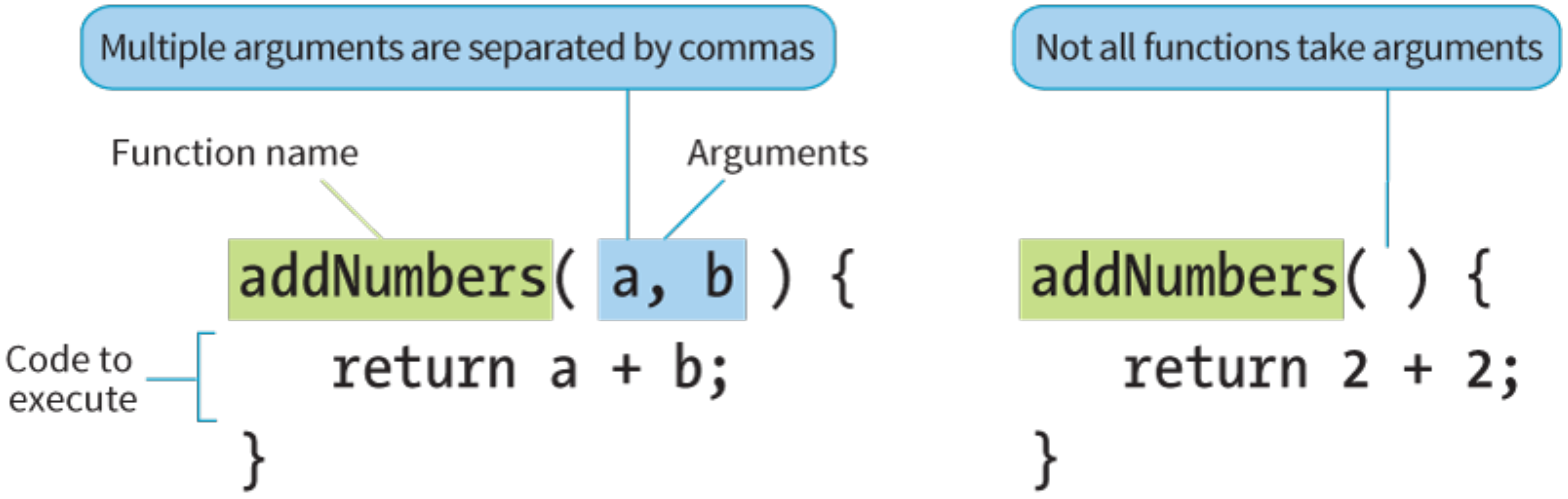
```
for(var i = 0, i <= 2, i++) {
   alert(i);
}
```

- **`for()`**: Says, "for every time this is true, do this."
- **`var i = 0`**: Creates a new variable **`i`** with its value set to 0. "i" (short for "index") is a common variable name.
- **`i <= 2`**: Says, "as long as **`i`** is less than or equal to 2, keep looping."
- **`i++`**: Shorthand for "every time this loop runs, add 1 to the value of **`i`**."
- **`{alert(i);}`**: This loop will run three times (once each for 0, 1, and 2 values) and alert the **`i`** value.

Functions

A **function** is a bit of code for performing a task that doesn't run until it is referenced or called.

Parentheses sometimes contain **arguments** (additional information used by the function):


Multiple arguments are separated by commas
Not all functions take arguments
Function name
Arguments
Code to execute

```
addNumbers( a, b ) {
    return a + b;
}
```

```
addNumbers( ) {
    return 2 + 2;
}
```

Functions (cont'd)

Some functions are built into JavaScript. Here are examples of **native functions**:

- **`alert()`**, **`confirm()`**, and **`prompt()`**
  Functions that trigger browser-level dialog boxes
- **`Date()`**
  Returns the current date and time

You can also create your own **custom functions** by typing **`function`** followed by a name for the function and the task it performs:

```
function name() {
   // Code for the new function goes here.
}
```

Variable Scope

A variable that can only be used within one function is **locally scoped**. When you define a variable inside a function, include the `var` keyword to keep it locally scoped (recommended):

```
var foo = "value";
```

A variable that can be used by any script on your page is said to be **globally scoped**.

- Any variable created *outside* of a function is automatically globally scoped:

```
var foo = "value";
```

- To make a variable created *inside* a function globally scoped, omit the `var` keyword:

```
foo = "value";
```

The Browser Object

JavaScript lets you manipulate parts of the browser window itself (the **`window`** object).

Examples of `window` properties and methods:

Property/Method	**Description**
`event`	Represents the state of an event
`history`	Contains the URLs the user has visited within a browser window
`location`	Gives read/write access to the URI in the address bar
`status`	Sets or returns the text in the status bar of the window
`alert()`	Displays an alert box with a specified message and an OK button
`close()`	Closes the current window
`confirm()`	Displays a dialog box with a specified message and an OK and a Cancel button
`focus()`	Sets focus on the current window

Event Handlers

An **event** is an action that can be detected with JavaScript and used to trigger scripts.

Events are identified by **event handlers**. Examples:

- **`onload`** When the page loads
- **`onclick`** When the mouse clicks an object
- **`onmouseover`** When the pointer is moved over an element
- **`onerror`** When an error occurs when the document or a resource loads

Event Handlers (cont'd)

Event handlers can be applied to items in pages in three ways:

- As an HTML attribute:

```
<body onclick="myFunction();">
/* myFunction runs when the user clicks anything
within 'body' */
```

- As a method attached to the element:

```
window.onclick = myFunction;
/* myFunction will run when the user clicks anything
within the browser window */
```

- Using **`addEventListener()`**:

```
window.addEventListener("click", myFunction);
```

Notice that we omit the preceding "on" from the event handler with this syntax.