JavaScript & Document Object Model (DOM)

The contents and slides of this topic are used with permission from:

- Jennifer Robbins, Learning Web Design, O'Reilly, 5th edition, May 2018, ISBN 978-1-491-96020-2.
- Paul S. Wang, Dynamic Web programming and HTML5, Routledge, 1 edition, 2012, ISBN 1439871825.

1

JavaScript

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

2

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.

3

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!

4

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>
```

5

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>
```

6

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.

7

Building Blocks of Scripts

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

8

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.

9

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.

10

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"`

11

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"`

12

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

13

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.
```

14

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

15

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.
▶ }
```

16

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!"

17

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
}
```

18

Loops (cont'd)

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

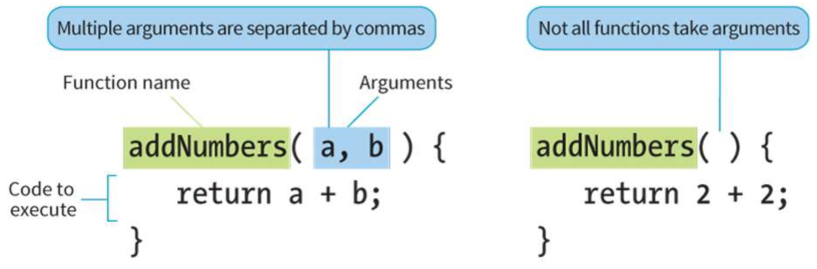
```
for(var i = 0, i <= 2, i++) {
   alert(i);
}
```

- `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.

19

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
Function name
Arguments
Code to execute
Not all functions take arguments

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

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

20

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.
▶ }
```

21

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";
```

22

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

23

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

24

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 */
```

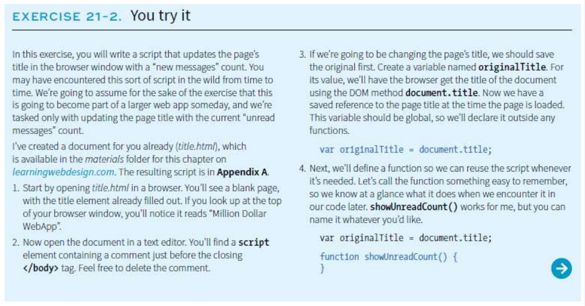
  - Using **`addEventListener()`**:

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

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

25

Exercise



EXERCISE 21-2. You try it

In this exercise, you will write a script that updates the page's title in the browser window with a "new messages" count. You may have encountered this sort of script in the wild from time to time. We're going to assume for the sake of the exercise that this is going to become part of a larger web app someday, and we're tasked only with updating the page title with the current "unread messages" count.

I've created a document for you already (*title.html*), which is available in the *materials* folder for this chapter on *learningwebdesign.com*. The resulting script is in **Appendix A**.

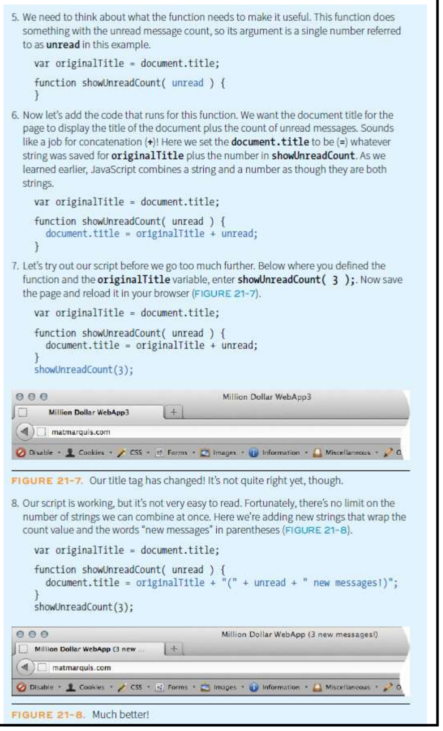
1. Start by opening *title.html* in a browser. You'll see a blank page, with the title element already filled out. If you look up at the top of your browser window, you'll notice it reads "Million Dollar WebApp".
2. Now open the document in a text editor. You'll find a **`script`** element containing a comment just before the closing **`</body>`** tag. Feel free to delete the comment.
3. If we're going to be changing the page's title, we should save the original first. Create a variable named **`originalTitle`**. For its value, we'll have the browser get the title of the document using the DOM method **`document.title`**. Now we have a saved reference to the page title at the time the page is loaded. This variable should be global, so we'll declare it outside any functions.

```
var originalTitle = document.title;
```

4. Next, we'll define a function so we can reuse the script whenever it's needed. Let's call the function something easy to remember, so we know at a glance what it does when we encounter it in our code later. **`showUnreadCount()`** works for me, but you can name it whatever you'd like.

```
var originalTitle = document.title;

function showUnreadCount() {
}
```



5. We need to think about what the function needs to make it useful. This function does something with the unread message count, so its argument is a single number referred to as **`unread`** in this example.

```
var originalTitle = document.title;

function showUnreadCount( unread ) {
}
```

6. Now let's add the code that runs for this function. We want the document title for the page to display the title of the document plus the count of unread messages. Sounds like a job for concatenation (**+**)! Here we set the **`document.title`** to be (**=**) whatever string was saved for **`originalTitle`** plus the number in **`showUnreadCount`**. As we learned earlier, JavaScript combines a string and a number as though they are both strings.

```
var originalTitle = document.title;

function showUnreadCount( unread ) {
  document.title = originalTitle + unread;
}
```

7. Let's try out our script before we go too much further. Below where you defined the function and the **`originalTitle`** variable, enter **`showUnreadCount( 3 );`**. Now save the page and reload it in your browser (FIGURE 21-7).

```
var originalTitle = document.title;

function showUnreadCount( unread ) {
  document.title = originalTitle + unread;
}
showUnreadCount(3);
```

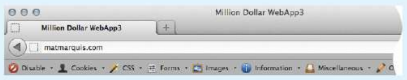


FIGURE 21-7. Our title tag has changed! It's not quite right yet, though.

8. Our script is working, but it's not very easy to read. Fortunately, there's no limit on the number of strings we can combine at once. Here we're adding new strings that wrap the count value and the words "new messages" in parentheses (FIGURE 21-8).

```
var originalTitle = document.title;

function showUnreadCount( unread ) {
  document.title = originalTitle + "(" + unread + " new messages!)";
}
showUnreadCount(3);
```

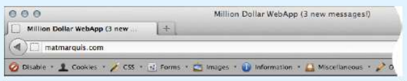

Million Dollar WebApp (3 new messages!)
Million Dollar WebApp (3 new ...
matmarquis.com
Disable
Cookies
CSS
Forms
Images
Information
Miscellaneous

FIGURE 21-8. Much better!

26

Document Object Model (DOM)

- **What the DOM is**
- **Accessing and changing elements, attributes, and contents**
- **Polyfills**
- **JavaScript libraries**

27

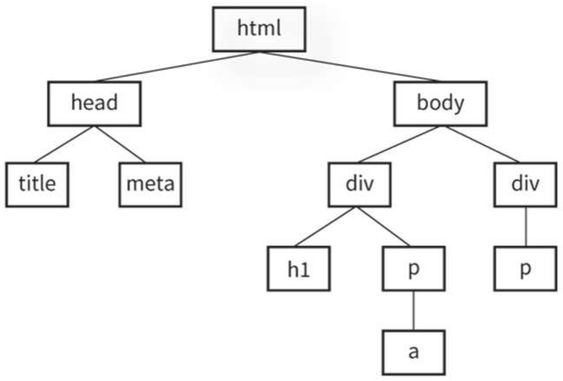
Intro to the DOM

- The **Document Object Model (DOM)** is a **programming interface** that provides a way to access and manipulate the contents of a document.
- It provides a structured **map of the document** and a set of **methods** for interacting with them.
- It can be used with other XML languages and it can be accessed by other programming languages (like PHP, Ruby, etc.).

28

Node Tree

- The DOM treats the structure of a document like a tree with branches:

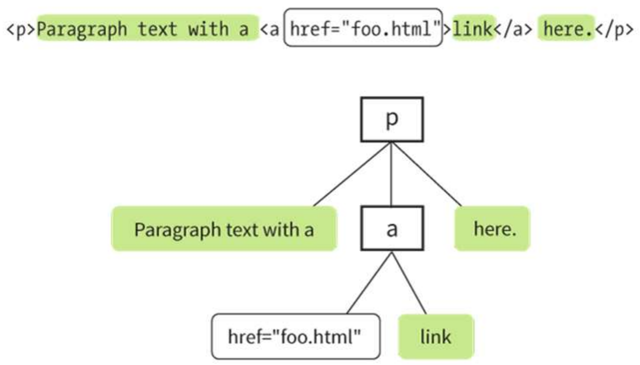


29

Node Tree (cont'd)

- Every element, attribute, and piece of content is a node on the tree and can be accessed for scripting:

The nodes within a `p` element


<p>Paragraph text with a <a href="foo.html">link</a> here.</p>
p
Paragraph text with a
a
here.
href="foo.html"
link

30

Accessing Nodes

- To point to nodes, list them separated by periods (.).
- In this example, the variable `foo` is set to the HTML content of an element with `id="beginner"`:
- `var foo = document.getElementById("beginner").innerHTML;`
- The **`document`** object points to the page itself.
- **`getElementById`** specifies an element with the `id` “beginner”.
- **`innerHTML`** stands for the HTML content within that element.

31

Accessing Nodes (cont’d)

- Methods for accessing nodes in the document:
- **`getElementsByTagName()`**
- Accesses all elements with the given tag name
    - **Example:** `document.getElementsByTagName("p");`
- **`getElementById()`**
- Accesses a single element by the value of its `id` attribute
    - **Example:** `document.getElementById("special");`
- **`getElementsByClassName()`**
- Access elements by the value of a class attribute
    - **Example:** `document.getElementsByClassName("product");`

32

Accessing Nodes (cont'd.)

- `querySelectorAll()`
- Accesses nodes based on a CSS selector
  - **Example:** `document.querySelectorAll(".sidebar p");`

- `getAttribute()`
- Accesses the value of a given attribute
  - **Example:** `getAttribute("src")`

33

Manipulating Nodes

- There are several built-in methods for manipulating nodes:

- `setAttribute()`
- Sets the value of a given attribute:
- `bigImage.setAttribute("src", "newimage.jpg");`

- `innerHTML`
- Specifies the content inside an element (including markup if needed):
- `introDiv.innerHTML = "<p>This is the intro text.</p>"`

- `style`
- Applies a style using CSS properties:
- `document.getElementById("intro").style.backgroundColor = "#000;"`

34

Adding and Removing Elements

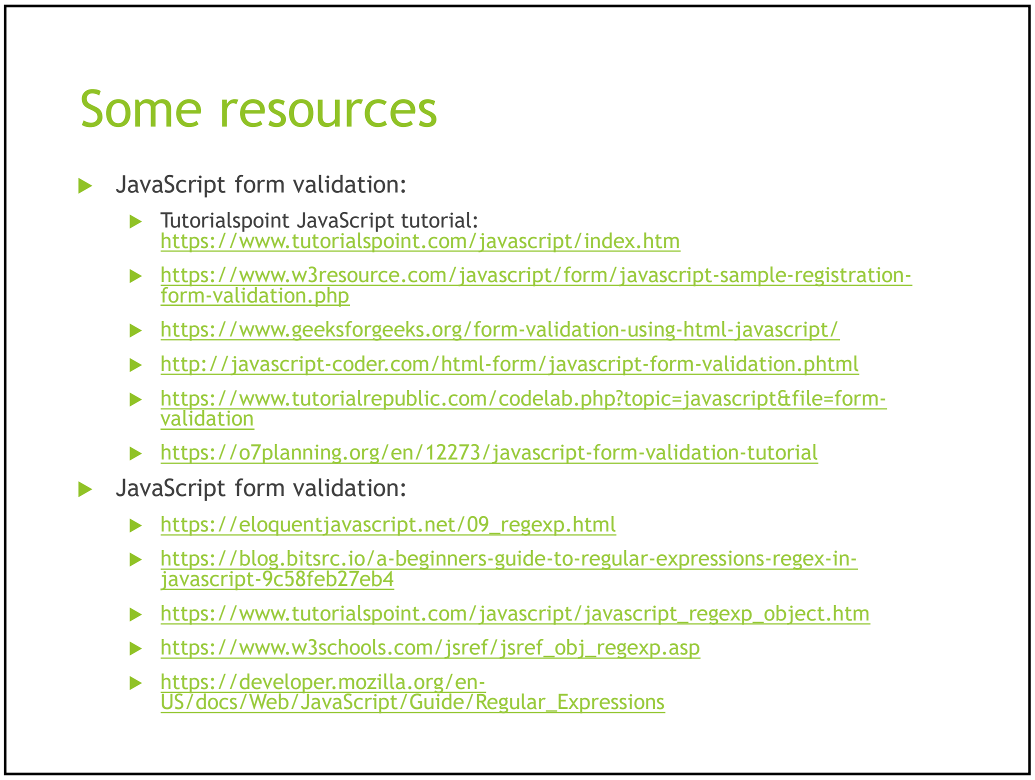
- The DOM allows developers to change the document structure by adding and removing nodes:
  - `createElement()`
  - `createTextNode()`
  - `appendChild()`
  - `insertBefore()`
  - `replaceChild()`
  - `removeChild()`

35

JavaScript Libraries

- A **JavaScript library** is a collection of prewritten functions and methods that you can use in your scripts to accomplish common tasks or simplify complex ones.
- Some are large frameworks for building complex applications.
- Some are targeted to specific tasks, such as forms or math.
- The most popular library is **jQuery**.
- Try searching "JavaScript library for ___________" to see if there are pre-made scripts you can use or adapt to your needs.

36



Some resources

- JavaScript form validation:
  - Tutorialspoint JavaScript tutorial: https://www.tutorialspoint.com/javascript/index.htm
  - https://www.w3resource.com/javascript/form/javascript-sample-registration-form-validation.php
  - https://www.geeksforgeeks.org/form-validation-using-html-javascript/
  - http://javascript-coder.com/html-form/javascript-form-validation.phtml
  - https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=form-validation
  - https://o7planning.org/en/12273/javascript-form-validation-tutorial
- JavaScript form validation:
  - https://eloquentjavascript.net/09_regexp.html
  - https://blog.bitsrc.io/a-beginners-guide-to-regular-expressions-regex-in-javascript-9c58feb27eb4
  - https://www.tutorialspoint.com/javascript/javascript_regexp_object.htm
  - https://www.w3schools.com/jsref/jsref_obj_regexp.asp
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

37