

Chapter 6: Javascript: Introduction to Scripting

CS 80: Internet Programming

Instructor: Mark Edmonds

Welcome to Javascript!

- What is javascript?
 - Client side scripting language
- What is scripting?
 - Generally tied with interpreted languages, meaning the code is translated to machine code right before execution (vs. compiled code)

Welcome to Javascript!

Why do we care?

- HTML and CSS give us a lot of control over content, structure, and presentation, but we still lack the ability to have dynamic or computational websites
- Javascript offers client-side programming, enabling fun stuff like these:
 - <https://codepen.io/HarrisCarney/pen/dPjKyK>
 - <https://codepen.io/GabbeV/pen/viAec>
 - <https://codepen.io/dissimulate/details/eZxEBO/>
- Clearly some powerful stuff!
- We will start small and with fundamentals; there are many libraries to help you with graphics
- We will concentrate on logic

Getting Started

- `<script>` HTML element allows us to add Javascript to our HTML document
 - Typically goes in the `<head>` portion of the document
- Example:

```
1 <script type="text/javascript"> script_stuff </script>
```

Example: hello_world.html

```
1 <!DOCTYPE html>
2 <!-- Fig. 6.1: welcome.html -->
3 <!-- Displaying a line of text. -->
4 <html>
5
6 <head>
7   <meta charset="utf-8">
8   <title>A First Program in JavaScript</title>
9   <script type="text/javascript">
10     // the document object accesses the current HTML document
11     // writeln is a method that writes a line to the document
12     // the enclosed argument to writeln is the content to write to the
        document
13     document.writeln("<h1>Welcome to JavaScript Programming!</h1>");
14   </script>
15 </head>
16
17 <body>
18   <!-- Notice our empty body -->
19 </body>
20
21 </html>
```

Example: hello_world.html

- Finally we see the object model!
- We will see more, but for now:
- `document` is the HTML document, represented through a javascript object
 - We will cover this concept in more detail in Chapter 12 (Document Object Model)
- `.writeln()` is a method within the `document` object
 - It writes content to the HTML document, followed by a new line (that's the `ln`)
 - `"<h1>Welcome to JavaScript Programming!</h1>"` is a **string** and is the content to write to the HTML document

External Javascript Files

- Example

```
1 <script src="srcipt.js">
```

- Loads the Javascript in `srcipt.js` into this HTML document

Javascript Basics

Statements and Keywords

- **Keywords** are words with special meaning to javascript
 - `var` is an example
- A **statement** is a program statement to execute by the javascript interpreter
 - Statements are terminated with a semicolon `;`, as in `hello_world.html`

Javascript Basics

Variables

- A **variable** is a container to store data in
 - Think about this like a math variable, but in javascript, variables can store any data
 - Declared with `var myVar;`
 - * Can declare multiple variables at once with a comma-serparated list
 - E.g. `var myVar, myVar1, myVar2;`
 - Assign the variable a value with `myVar = 5;`

Example: `hello_world_variables.html`

```
1 <!DOCTYPE html>
2 <!-- Fig. 6.1: welcome.html -->
3 <!-- Displaying a line of text. -->
4 <html>
5
6 <head>
7   <meta charset="utf-8">
8   <title>A First Program in JavaScript</title>
9   <script type="text/javascript">
10     // declare a variable to hold our string
11     var text = "<h1>Welcome to JavaScript Programming!</h1>";
12     var myDocVar = document;
13
```

```
14     // use the variables to write the content to the document
15     myDocVar.writeln(text);
16 </script>
17 </head>
18
19 <body>
20     <!-- Notice our empty body -->
21 </body>
22
23 </html>
```

Javascript Basics

Variables

- Sidenote: Javascript variables do not have a type; their type is determined by the content they store
 - What does this mean? Well, we've seen `var` be assigned to a string (`hello_world_variables.html`) and to a number (above, `myVar = 5;`).
 - * So `var` represents any variable, could be a string, could be an int
 - * This contrasts with many programming languages, like C/C++, Java
 - * Moreover, a variable is not bound to a particular type. We can reassign it at any time to any type (e.g. `myVar = 5; myVar = "cheese";`)

Javascript Basics

Variables

- Despite the fact variables can store any types, we still have to have a notion of types (take more CS courses to learn more about this!)
 - **String:** a string of text. E.g. `var myVar = "cheese";`
 - * Can use double or single quotes, but must be consistent (whatever you start the string with, you must end the string with)
 - **Number:** a number. E.g. `var myVar = 5;`
 - **Boolean:** True/False value. E.g. `var myVar = true;`

Javascript Basics

Variables

- **Array:** multiple of values in one variable. E.g. `var myVar = [5, "cheese", false];`
 - Elements are accessed using 0-indexing
 - E.g. `myVar[0]` is 5, `myVar[2]` is **false**
- **Object:** everything in javascript is an object, and you can store objects in variables. E.g. `var myVar = document;`

Javascript Basics

- **Identifiers**
 - Formal name for a variable's name (e.g. `myVar`)
 - Can contain letters, digits, underscores, and dollar signs.
 - Must not begin with a digit and must not be a keyword

Javascript Basics

- Comments
 - Single line: start with `//`
 - Multiline: start with `/*` ends with `*/`

Javascript Basics

- **Literals** are literal values you provide your script
 - They do not change value
 - Can you spot the literal?
 - * `var myVar = 5;`
 - * `var myVar = "cheese";`
 - These are not modifiable, fixed values provided by you, the programmer.

Javascript Basics

Basic Operators

- **Basic Operators** define operations on variables or literals. Used to process data. We will talk about more as we go along.

Javascript Basics

Basic Operators

- **Addition/Concatenation:** +, used to add two numbers together, or merge two strings
 - `6 + 9`; yields 15
 - `"Hello " + "world!"` yields `"Hello world!"`

Javascript Basics

Basic Operators

- **Subtract, multiply, divide:** -, *, /, used just as they in basic math (you can't divide a string by another string, concatenation is special!)
 - `6 * 5`; yields 30

Javascript Basics

Basic Operators

- **Assignment:** =, we've already seen this. It's used to take a value (either an object or a literal) and assign it another object (typically a variable)
 - `var myVar = 6 * 5`; assigns `myVar` to 30

Javascript Basics

Basic Operators

- **Remainder:** %, used to perform modulo.
 - Modulo/remainder division finds the remainder of an integer division
 - E.g. `11 % 5` yields 1 because `11 = 5 * 2 + 1`
 - * We are interested in the 1, since that is the remainder when you divide 11 by 5

Javascript Basics

Operator Precedence

- Remember PEMDAS? (Parenthesis, Exponents, Multiplication and Division, and Addition and Subtraction)
 - Defines the mathematical order-of-operations

Javascript Basics

Escape Sequences

- Used to give or take away meaning from special characters
- `\n`: a new line (carriage return, like hitting enter)
- `\t`: a tab
- `\\`: a literal backslash (since `\` normally has special meaning - to escape other characters!)
- `\"`: double quote - for nested double quotes
 - Need to escape so we don't accidentally end the string!
- `'`: single quote - for nested single quotes
 - Need to escape so we don't accidentally end the string!

Javascript Basics

Examples

- `alert.html`
 - `window` object refers the browser's window
 - `alert` opens a dialog to display the string
 - Take a look at `window` methods and attributes: http://www.w3schools.com/jsref/obj_window.asp

Example: `alert.html`

```
1 <!DOCTYPE html>
2 <!-- Fig. 6.3: welcome3.html -->
3 <!-- Alert dialog displaying multiple lines. -->
4 <html>
5
6 <head>
7   <meta charset="utf-8">
8   <title>Printing Multiple Lines in a Dialog Box</title>
9   <script type="text/javascript">
10     // Your book does the following <!-- // --> pattern inside the
       script tag all the time
11     // DONT DO THIS! It's for ANCIENT browsers that do not support
       scripts (and interprets them as an HTML comment)
12     // Seriously, I cannot stress how pointless this practice is, and
       you should not be doing it
```

```
13     <!--
14     window.alert("Welcome to\nJavaScript\nProgramming!");
15     // -->
16 </script>
17 </head>
18
19 <body>
20   <p>Click Refresh (or Reload) to run this script again.</p>
21 </body>
22
23 </html>
```

Javascript Basics

Examples

- `dynamic_welcome.html`
 - Again using `window` with `prompt()` method to ask for user input
 - Creates a dynamic welcome page
 - We couldn't do with this HTML and CSS

Example: `dynamic_welcome.html`

```
1 <!DOCTYPE html>
2 <!-- Fig. 6.5: welcome4.html -->
3 <!-- Prompt box used on a welcome screen -->
4 <html>
5
6 <head>
7   <meta charset="utf-8">
8   <title>Using Prompt and Alert Boxes</title>
9   <script type="text/javascript">
10     var name;
11     // string entered by the user
12     // read the name from the prompt box as a string
13     name = window.prompt("Please enter your name");
14     document.writeln("<h1>Hello " + name + ", welcome to JavaScript
15                       programming!</h1>");
16   </script>
17 </head>
```



```
17
18 <body></body>
19
20 </html>
```

Javascript Basics

Examples

- `addition.html`
 - `parseInt()` function converts a string to an integer

Example: `addition.html`

```
1 <!DOCTYPE html>
2 <!-- Fig. 6.7: addition.html -->
3 <!-- Addition script. -->
4 <html>
5
6 <head>
7   <meta charset="utf-8">
8   <title>An Addition Program</title>
9   <script type="text/javascript">
10    <!--
11     var firstNumber; // first string entered by user
12     var secondNumber; // second string entered by user
13     var number1; // first number to add
14     var number2; // second number to add
15     var sum; // sum of number1 and number2
16     // read in first number from user as a string
17     firstNumber = window.prompt("Enter first integer");
18     // read in second number from user as a string
19     secondNumber = window.prompt("Enter second integer");
20     // convert numbers from strings to integers
21     number1 = parseInt(firstNumber);
22     number2 = parseInt(secondNumber);
23     sum = number1 + number2; // add the numbers
24 //    sum = firstNumber + secondNumber; // add the numbers (as strings)
25     // display the results
26     document.writeln("<h1>The sum is " + sum + "</h1>");
27     // -->
```

```
28   </script>
29 </head>
30
31 <body></body>
32
33 </html>
```

Javascript Basics

Operators and Conditionals

- We need a way to encode logic - a way to direct the program based on the program's state
- Primary method of controlling a programs flow
- Basic idea: "if a condition is true, execute some code"
 - We can stop at the if or say "if a condition is true, execute some code, otherwise, execute this other code"

Javascript Basics

Operators and Conditionals

- Example:

```
1 // basic conditional
2 if(5 <= 10){
3   document.writeln("5 is indeed less than 10");
4 } else {
5   document.writeln("5 is somehow not less than 10...");
6 }
```

Javascript Basics

Operators and Conditionals

- But we can do even more by nesting if's!
 - Also allows us to check for multiple potential conditions at once (e.g. `if(cond) ...else if(cond) ...else...`)

Javascript Basics

Operators and Conditionals

- Example: where `time` is the hour of the day (0-23)

```

1 // order of the conditionals matters; what happens if we flip them
  ?
2 if (time < 10) {
3   greeting = "Good morning";
4 } else if (time < 20) {
5   greeting = "Good day";
6 } else {
7   greeting = "Good evening";
8 }

```

Javascript Basics

Equality and Relational Operators

Standard algebraic equality operator or relational operator	JavaScript equality or relational operator	Sample JavaScript condition	Meaning of JavaScript condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Figure 1: Javascript relational operators

Javascript Basics

Operators and Conditionals

- Difference between `===` and `==`
 - `===` is a more strict comparison operator
 - E.g. `"75" == 75` yields true but `"75" === 75` yields false

Javascript Basics

Operators and Conditionals

- Compound conditionals:
 - Use logical operations `AND`, `OR`, and `NOT`
 - `AND` is represented with `&&`
 - `OR` is represented with `||`
 - `NOT` is represented with `!`
 - Combining conditions allows us to use much more powerful program flow

Javascript Basics

Operators and Conditionals

- Compound conditionals will evaluate left to right
 - If the program can determine the overall value of the compound conditional, it will stop evaluating the rest of the conditional
 - E.g. Suppose `cond_a` is `true` and `cond_b` is `false`.
 - * `if(cond_a || cond_b)` doesn't need to look at the value of `cond_b`, the overall condition is determined by `cond_a`.
 - * Similarly, `if(cond_b cond_a)` doesn't need to look at the value of `cond_a`, since `cond_b` already determined the overall state of the compound conditional.
 - * Remember: left to right!
 - **Key takeaway** order matters!

Javascript Basics

Operators and Conditionals

- Example: (pay close attention to evaluation order)

```
1 // be careful analyzing this conditional
2 if(cond_a && cond_b){
3     // only executes if cond_a AND cond_b are true
4 } else if(cond_a || cond_d){
```

```
5 // only executes if 1) cond_a is true AND cond_b is false (think
   about why) OR cond_d is true
6 } else if(!cond_d){
7 // only executes if cond_a is false AND cond_d is true
8 } else {
9 // will this ever execute?
10 // otherwise
11 }
```

Exercise

- Display the current day and time in the following format:

```
1 Today is: Friday
2 Current time is: 4:50:22PM
```

- **First step:** look at the `Date()` javascript function
 - http://www.w3schools.com/jsref/jsref_obj_date.asp

Example: date_print.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <title>Display Date</title>
7   <script type="text/javascript">
8     var today = new Date();
9     var day = today.getDay();
10    var daylist = ["Sunday", "Monday", "Tuesday", "Wednesday ", "
        Thursday", "Friday", "Saturday"];
11
12    document.writeln("Today is: " + daylist[day] + "<br>");
13
14    var hour = today.getHours();
15    var minute = today.getMinutes();
16    var second = today.getSeconds();
17    console.log("hour: " + hour);
18    var postpand = "";
```

```
19     if (hour >= 12) {
20         postpand = "PM";
21     } else {
22         postpand = "AM";
23     }
24     if (hour >= 12) {
25         hour = hour - 12;
26     }
27     if (minute < 10) {
28         minute = "0" + minute;
29     }
30     if (second < 10) {
31         second = "0" + second;
32     }
33     document.writeln("Current time is: " + hour + ":" + minute + ":" +
34         second + postpand);
35 </script>
36 </head>
37 <body>
38 </body>
39
40 </html>
```

Exercise

- Write a program that determines if the inputted year is a leap year:
 - Every year that is divisible by 4, except for years divisible by 100, except for years that are divisible by 400
- **First step:** come up with an algorithm to check

Example: leap_year_checker.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8">
6     <title>Leap Year Checker</title>
7     <script type="text/javascript">
```

```
8     var year, leapYear = false,
9         leapYearText = "";
10    year = parseInt(window.prompt("Input a year: "));
11    if (year % 100 === 0) {
12        if (year % 400 === 0) {
13            leapYear = true;
14        }
15    } else if (year % 4 === 0) {
16        leapYear = true;
17    }
18    if (!leapYear) {
19        leapYearText = " not";
20    }
21    document.writeln("<h2>" + year + " is" + leapYearText + " a leap
22                      year</h2>");
23 </script>
24 </head>
25 <body>
26 </body>
27
28 </html>
```

Developer Tools

- No program is perfect at first
- We need a way to examine program state as the program is running
 - This will enable us to understand and fix problems in our code
- Debugging will help fix any error, but it's particularly good for identifying edge cases you may have not considered when writing your program
 - Common edge cases: empty string "", zero 0, negative numbers, etc

Developer Tools

- The debugger is present in any modern browser and has two main elements for javascript: console and debugger
- **Since we are editing the program as it executes, we need to remember where we are in execution**
 - Variables may not exist yet

- How to write information to the console to inspect state:
 - Use a `console.log()` statement in your javascript
 - The value/text arguments will be printed to the Console section of the Developer Tools

Developer Tools

- Setting a breakpoint
 - Breakpoints say "when you hit this line of code, pause the program for me"
 - They allow you to inspect variable/program state during execution
 - Enable a breakpoint by clicking the line number
 - Important semantic note: The line of the breakpoint has **not** executed yet. It is about to execute
 - * E.g. if we set a breakpoint on line 9, line 9 hasn't executed when the breakpoint triggers (line 8 has, however)
 - * Keep this in mind!

Developer Tools

- Stepping through a program
 - Once we are at a breakpoint we have multiple options to control the program:
 - * Step Over: step over the current line of code. This means if we are at a function call, **do not** move the debugger into the function. Instead, the function will execute, and the program will pause after function completes
 - * Step Into: If the program is about to call a function, move the debugger into that function and pause execution
 - * Step Out: Finish the current function call and pause execution at the calling function
 - * Continue: Continue the program's execution; basically unpauses the program

Developer Tools

- We can also call functions or inspect variables while paused using the console
 - Move to the console, and type a javascript statement to execute
 - * Can also just type variable names to get their value
 - If the javascript statement we called doesn't have a return value (e.g. statement doesn't yeild a value), then the console will report `undefined`.

Javascript Functions

- Functions enable reusable code
- Sum example:

```
1 // function declaration
2 function sum1(a, b){
3   // do other amazing javascript things here
4   return a + b;
5 }
```

- We can then call `sum(10, 5)` which would return 15

Javascript Functions

- Assigning functions to variables
- Sum example:

```
1 // anonymous function assigned to variable
2 var sum2 = function(a,b){
3   return a + b;
4 }
```

- We can then call `sum2(10, 5)` which would return 15

Javascript Functions

Hoisting

- The only thing you need to remember is that functions of the form of `sum1` are "hoisted" to the top of your program, meaning they can be used *before* they are declared