Intro to Programming

(Especially JavaScript, but also kinda PHP)

Summary

- What is programming?
- Meet JavaScript
- Basic syntax, creating a program
- Programming foundations
  - Variables, data types, conditionals, loops, functions
- Tools and resources

Why a full review of JavaScript?

It may seem silly to re-learn JS if you already know it, or right after foundations, but there are several good reasons:

- Review is always good for learning
- JavaScript is important in any type of web development, and has never been bigger
- You might learn something new you didn't know about
- **JavaScript and PHP are extremely similar**

1. What is programming?

Baby, don't hurt me
Don't hurt me
No more

What is programming?

- Giving direct instructions to a computer
- Breaking complex problems into simple steps
- Using the right language for the problem
- Using the language's documentation
- Coding the proper statements, in the proper order

Programming languages

- Your computer only understands machine code (zeroes and ones)
- Machine code isn't very writable, or readable
- All programming languages are trying to bridge the gap between humans and the machine code
- They're kind of like compromises

Programming languages

- There are hundreds of programming languages
- All at varying levels of relation to machine code
- Low-level languages (closer to machine code), high-level languages (closer to human language)
- We write **source** code - but it all **compiles** to machine code
- All languages have a combination of characters and symbols that define the language's syntax

Compiled vs. interpreted languages

- Some code uses a compiler, which creates an executable
- Executable is read, and the source code is hidden
- Some code is read directly by an interpreter
- **Browsers are HTML, CSS & JS interpreters**
- You can view source on HTML, CSS, and JS resources; the code doesn't need to be compiled. It's sent to the browser directly as-is.
- **PHP is an interpreted language**, but it's interpreted on the server rather than in the browser. More on that later...

2. Meet JavaScript

No Java, but plenty of Script

JavaScript

An object-oriented computer programming language commonly used to create interactive effects within web browsers.

Why JavaScript?

- Maybe not the most powerful or flexible (though much more so than HTML/CSS)
- But definitely relevant and popular
- It's grown exponentially in the last 5–10 years
- Very practical for working with web pages; it’s built into every browser out there
- Intentionally limited (can’t ruin things too badly)
- Huge community, long history

JavaScript History

- Prototype developed by Brendan Eich of Netscape in '95 (in 10 days!)
- Originally called Mocha, then LiveScript before eventually JavaScript.
- Also known as ECMAScript, after ECMA (European Computer Manufacturers Association), which oversees its standardization
- **JavaScript has nothing to do with the language Java**; its similar naming was a marketing decision
- Script: instructions a computer can run one line at a time

JavaScript History

- 1996 - Netscape 2 is released with JavaScript support
- 1997 - First standardized version of JavaScript
- Early 2000s - JavaScript development slows to a crawl as browsers fail to agree on standards and improvements to the language
- 2005 - AJAX is first coined. AJAX is used to communicate with servers asynchronously (without requiring a reload, in real-time) using JavaScript.
- 2006 - jQuery, a super-popular JavaScript library, debuts.
- 2009 - After a long period of disagreement between parties involved, ES5 is released to provide a more standardized version of JavaScript

JavaScript History

- 2010 - Node.js debuts. This provided a way to develop server-side and networking applications in JavaScript
- 2015 - ES6 was released, and began adoption into modern browsers.
- TODAY - Not all features of ES6 are fully supported yet, but most are (or are close). ES7 has new features already shipping, too!
  - JavaScript is all the hotness
  - Full-fledged apps are built with it
  - You can't throw a stone without hitting a JS library/framework.

Including JavaScript

Much like including CSS!

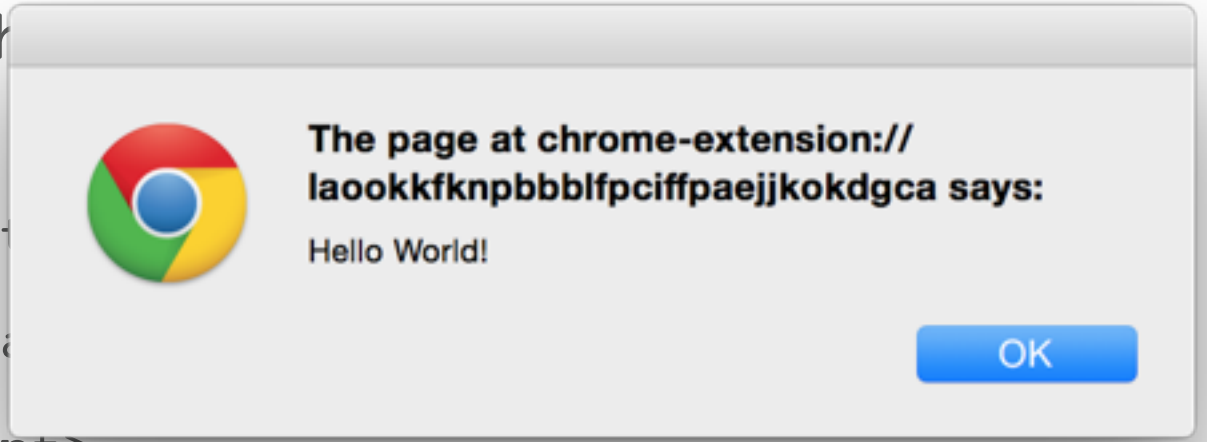
Including JavaScript in HTML

You can write a “script” tag in your HTML document to write the JavaScript right there:

```
<script>
      alert('Hello World!');
</script>
```

Including JavaScript in HTML

You can write a “script” tag in your HTML document to write th




```
<script
        a
</script>
```

Including JavaScript in HTML

OR you can link to a JavaScript file from the head of your HTML document, just like linking an external CSS stylesheet:

```
<head>
        <script src="myscripts.js"></script>
</head>
```

Event Handlers

- JavaScript within HTML - not script tags
- Execute when something happens
- onClick, onLoad, onMouseover, onMouseout, etc
- jQuery is capable of including many more event handlers than are available by default in HTML

- ```
  <a href="http://google.com"
  onClick="alert('Hello!')">Google</a>
  ```
  ```

Event Handlers

- There are also event listeners in JavaScript
- These are added globally with JS, rather than to HTML
- We'll touch on these more later; don't worry about them for right now

```
button.addEventListener('click', myFunction)
```

Wait, what's with the weird capitalization?

- JavaScript generally capitalizes every word **except** the first one (though there are a few exceptions)
- **This is called "camel case"**
- Example: `document.getElementsByTagName();`
- It's generally encouraged to follow this naming convention when creating your own names for things (more on that later), but not strictly necessary

Basic functions to play with

```
document.write();
//Used to “print” text to your web page
console.log();
//Log unseen data to the console for debugging
alert();
//Creates a popup. Use sparingly! (Or never.)
prompt();
//Creates a popup asking for information or confirmation
```

NOTE!

Browsers run code the moment they come across it.

In the previous example, the JavaScript to “alert” the user is read—and executed—before any page content can be loaded. The browser “pauses” to run the code. This is an important concept in bigger sites and applications.

NOTE!

- It can be easy to ruin code in JS with a typo.
- Only one of the following works!

```
document.getElementById("main").style.display = "block";

document.getElementByID("main").style.display = "block";

document.getElementsById("main").style.display = "block";

document.GetElementsById("main").style.display = "block";
```

JavaScript is case-sensitive.

That's important, so it gets this slide.

Including External Files

```
<!-- HTML4 and (x)HTML -->

<script type="text/javascript" src="scripts.js">


<!-- HTML5 -->

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

3. Basic Syntax & Writing a Program

Now we're cooking with fire!

Writing Text on a Web Page

- There are many techniques to accomplish this
- To start, we'll use a built-in JavaScript command

```
<script>
      document.write('Hello, World!');
</script>
```

Writing Text on a Web Page

- There are many techniques to accomplish this
- To start, we'll use a built-in JavaScript command

```
<script>
    document.write('<h1>Hello, World!</h1>');
</script>
```

Comments

- // Single line comment

```
var price = 10;
// You can add comments after statements


/*
This is a multi-line comment
*/
```

Comments in external libraries

```
/*
 * jQuery JavaScript Library v1.11.2
 * http://jquery.com/
 *
 * Copyright 2005, 2014 jQuery Foundation, Inc. and
other contributors
 * Released under the MIT license
 * http://jquery.org/license
 *
 * Date: 2014-12-17T15:27Z
 */
```

4. Programming Foundations

Get ready to do some awesome stuff!

Values

- Any piece of data that a computer program can store and use
- There are several data types in JavaScript, but there are four main simple types:
  - **String** – for textual values like `"Flywheel"`
  - **Number** – for numeric values like `1405`
  - **Array** – lists, like `["Red", "Orange", "Yellow", "Blue"]`
  - **Boolean** – `true` or `false` (also sometimes called `1` or `0`)

Values

```
 347  // Integer, or number; can involve a decimal
 "Hello, World!"  // String
 [2.43, 97.60, 9.25]   // Arrays; can mix types
["Red", "Orange", "Yellow"]
 true  // Boolean
 null // Means “no value”
 NaN  // Not a Number
 Undefined // Existent but with unassigned value.
```

Variables

- A storage location for a value your program can use
- The value can vary
- Think of a variable a little like a basket
- Much like algebra, variables hold a piece of data
- Also like algebra, we can make operations with variables, e.g., item + shipping = total

Variables

- **Declare** a variable = give the value a name

```
var cookies;
```

- **Initialize** variable = give it a value

```
cookies = 5;
```

- **Declare AND initialize** at the same time

```
var cookies = 5;
```

- **Change** value of variable

```
cookies = 4; //I ate a cookie
```

Naming Variables

- A variable's name can only start with a letter, `$` or `_`
- Must only contain letters, numbers, `$` and `_`
  - **No spaces or special characters**
- Variable names are case-sensitive
  - `myvariable` is different than `myVariable`
- Avoid keywords already used in JavaScript to mean something:
  - `var, alert, document, window, new, this, function, if, else, case, break, etc.`

Naming Best Practice

- In programming, we get to decide our own names for things pretty frequently
- There's a good way and a bad way to name things...

Naming Best Practice

- **DO:**
  - Choose a descriptive name for data
  - Choose a name that isn't overly lengthy
  - Avoid potential conflicts with other code (use unique identifiers)
- **DON'T:**
  - Choose non-descriptive names
  - Choose names that are extremely generic
  - Use keywords or other names close to existing code

Naming Best Practice

- DO:
  - `var userName; //descriptive`
  - `var userCity; //descriptive`
  - `var et_pb_user; //descriptive and unique`
- DON'T:
  - `var myVariable; //nondescript`
  - `var m; //extremely nondescript`
  - `var theUserNameAssociatedWithTheCurrentUser; //too long`
  - `var string; //confusing to us and to JavaScript`

Once again, values

- Any piece of data that a computer program can store and use. Variables are frequently used for just this purpose.
- Again, the four **main** simple data types:
  - **String** – for textual values like `"Flywheel"`
  - **Number** – for numeric values like `1405`
  - **Array** – lists, like `["Red", "Orange", "Yellow", "Blue"]`
  - **Boolean** – `true` or `false` (also sometimes called `1` or `0`)

String

- A text value
- Can be any kind or amount of text, from a single character to many paragraphs.
- **Always** surrounded by quotes, so that JavaScript can tell when the string begins and ends.

```
var myString = "What's up, peeps?"
```

Side Note on Quotes:

- You can use single or double quotes anywhere quotes are called for
- The only rule is that you must start and end with the same character
- There's no difference, but some have personal preferences
- **Beware auto formatting!** Curly quotes don't work in code

```
var myString = "Hi there!";        //Works fine; starts and ends with "
var myString = 'Hi there!';        //Also works; starts/ends with '
var myString = "What's up?";       //Still works fine
var myString = 'What's up?';       //Oops!
var myString = “What’s up?”;       //Very bad. Sad.
```

Escaping Strings

- You can "escape" a character with a backslash
- This tells JavaScript to treat it like part of the string rather than as code

```
var myString = 'What's up?';     //Error!
var myString = 'What\'s up?';    //"What's Up?"
```

Combining Strings

- Known as "**concatenation**"
- Uses the + operator, which "adds" the strings, but not in mathematical terms; it just combines them.
- Common programming task

```
var firstName = 'Josh';

var lastName = 'Collinsworth';

var fullName = firstName + lastName;
```

Combining Strings

- Known as "**concatenation**"
- Uses the + operator, which "adds" the strings, but not in mathematical terms; it just combines them.
- Common programming task

```
var firstName = 'Josh';
var lastName = 'Collinsworth';
var fullName = firstName + ' ' + lastName;
```

Using Variables

- Just type the variable name
  - `alert(apples);`
- **Don't use quotes with variables**; use quotes for strings
- `alert(apples);` is different than `alert('apples');`
  - The first is the **variable** apples; the second is the **string** "apples"
- **Without quotes,** the JavaScript Interpreter treats words as special JavaScript objects, or as variable names.
- **With quotes,** they'll just be read as plain text.

New in ES6: Template Strings

- Also called “template literals”
- Allows variables within strings; no concatenation!
- Just use backticks instead of quotes, and place variables in `${ here }`
- Here’s how you’d have to do it the old way:

```
var firstName = 'Josh';

var lastName = 'Collinsworth';

var greet = 'Welcome, ' + firstName + ' ' + lastName + '!';

//Welcome, Josh Collinsworth!
```

New in ES6: Template Strings

- Also called “template literals”
- Allows variables within strings; no concatenation!
- Just use backticks instead of quotes, and place variables in `${ here }`
- **Here's the new way:**

```
var firstName = 'Josh';

var lastName = 'Collinsworth';

var greet = `Welcome, ${firstName} ${lastName}!`;

//Welcome, Josh Collinsworth!
```

ES6 Template Strings

- Template strings also allow line breaks in strings! **Here's the old way:**

```
var lyrics = "What is love?\
    Baby, don't hurt me\
    Don't hurt me\
    No more";
```

- **Or this:**

```
var lyrics = "What is love?" + "Baby, don't hurt me" +
    "Don't hurt me" + "No more";
```

ES6 Template Strings

- Template strings also allow line breaks in strings!
- **Here's the new way (just use backticks):**

```
var lyrics = `What is love?

    Baby, don't hurt me

    Don't hurt me

    No more`;
```

New in ES6: Template Strings

- Template strings might not always be easier
- But they really shine when you need to combine a variable with a complex string, like a URL

```
var siteDomain = 'http://mysite.com';

contactPage = `${siteDomain}/about/contact`;

//comes out as: http://mysite.com/about/contact
```

Number

- A numerical value
- Can be whole number or decimal
- **Some languages distinguish between types of numbers** (decimal, whole, short, long). JS doesn't; a number is a number to JavaScript.

```
var myNumber = 12;
var myNumber = 1.61803;
```

Remember: No Quotes with Numbers!

- JavaScript knows whether it's dealing with a string or a number by the quotes (or lack thereof).

```
var myFirstNumber = 42;

var mySecondNumber = "42";


typeof(myFirstNumber)     //"number"

typeof(mySecondNumber)    //"string"
```

Basic Math

- JavaScript supports addition, division, subtraction, etc. Operators:

```
+       // Adds two numbers
-       // Subtracts one number from the other
*       // Multiplies two numbers
/       // Divides one number by another
++      //  Increment (add one to the number)
--      //  Decrement (subtract one from the number)
%       //  Modulus; divides two numbers & returns remainder
```

Basic Math

- JavaScript supports addition, division, subtraction, etc. Operators:

```
+       //  12 + 7 =  19
-       //  12 - 7 =  5
*       //  12 * 7 =  84
/       //  12 / 7 =  1.714285714
++      //  12++ =    13
--      //  12-- =    11
%       //  12 % 5 =  2
```

Basic Math

- ```
  var price = 10;
  // item price is 10
  ```
- ```
  var itemsOrdered = 15;
  // items ordered is 15
  ```
- ```
  var totalCost = price * itemsOrdered;
  // total cost is 10 * 15, or 150
  ```

- **Don’t forget the order of operations!**
  ```

Quick order of operations refresher

- PEMDAS - “Please Excuse My Dear Aunt Sally”
- Parentheses, Exponents, Multiplication/Division, Addition/Subtraction
- `((10 * 2) / 5) + 2 * 3` `//Parentheses first`
- `(20 / 5) + 2 * 3` `//Work inward out`
- `4 + 2 * 3` `//Multiplication/Division`
- `4 + 6` `//Addition/Subtraction`
- `10` `//Result, victory, parties!`

General Comparison Operators

- Used to compare two values

```
==      // Equal to (note the double "=")
>       // Greater than
>=      // Greater than or equal to
<       // Less than
<=      // Less than or equal to
```

Basic Comparison Operators

Compare:	True or false?
5 == 5	`//True`
5 == 2	`//False`
12 > 10	`//True`
12 >= 10	`//True`
7 < 7	`//False`
15 <= 15	`//True`

The "not" operator

- Used to invert the "truthiness" of a statement

- `==  // Equal to`
- `!=  // NOT equal to`

Basic Comparison Operators

- Note that comparison operators can be used on strings as well as numbers.

Compare:	True or false?
5 == 5	`//True`
5 != 5	`//False`
12 == 10	`//False`
15 != 9	`//True`
"true" != "false"	`//True`
"alternative facts" == "facts"	`//False`

Type Conversion

- **By default when we use the == comparison operator, JavaScript uses "type conversion"**
- This means if we try to compare two values of different **types**, JavaScript will help us out a little bit
- `"2" == 2      //true, even though it's a string and number`
- If we want to **avoid type conversion**, we need to use === to check for value **and** type

Equal Value AND Type

- Used to check whether the data matches AND the type of data matches

- `==  // Equal values`
- `===  // Absolutely equal values AND data types`
- `!=  // Unequal values`
- `!==  // NOT absolutely equal`

Basic Comparison Operators

Compare:	True or false?
"5" == 5	`//True`
"5" === 5	`//False`
"5" != 5	`//False`
"5" !== 5	`//True`

Comparison Operators

- Don’t confuse = with == or ===
- The first **assigns** a value; the others **compare** values.

```
var firstName = "Josh";

// Create the variable firstName and assign it the value of
“Josh”


firstName == "Josh";

// check to see if firstName equals "Josh" (true or false)
```

Logic

- JavaScript gives us some logical operators for common conjunctions such as "and", and "or"
- Operators:

```
&&          // and
||          // or
!           // not
```

Logic

```
var apples = 5;

var oranges = 3;


console.log(apples > 4 && oranges > 4);   //false

console.log(apples > 4 || oranges > 4);   //true

console.log(apples === oranges);          //false

console.log(apples !== oranges);          //true
```

The IF Statement

- Execute code based on conditions
- Code inside curly brackets runs if the statement is true, but does nothing if the statement is not true.
- **Extremely important!** You will use these in JavaScript and in PHP later on, so learn them well! (The syntax is identical.)

```
if (condition) {
        // statement to execute
}
```

The IF Statement

```
if (condition) {
      effect
}
```

```
if (this is true) {
      then do this
}
```

The IF Statement

- Execute code based on conditions

```
var apples = 1;

if (apples < 2) {
    console.log("You should buy apples.");
}
```

IF/ELSE Statement

- Use “else” to perform an alternative action if first condition fails

```
var apples = 5;


if (apples > 3) {
    console.log('Eat an apple!');
} else {
    console.log('Buy an apple!');
}
```

IF/ELSE Statement

```
if (age >= 35) {
        console.log('You can vote and hold any office.');
} else if (age >= 30) {
        console.log('You can vote, and run for the Senate.');
} else if (age >= 25) {
        console.log('You can vote, and run for Representative.');
} else if (age >= 18) {
        console.log('You can vote.');
} else {
        console.log('You have no voice in government (yet).');
}
```

Bad data types

- So far, we've covered strings and numbers
- You may run into these other data types when a program doesn't go as expected:

```
NaN

Undefined

Null
```

NaN

- NaN = Not a Number. Often the result of mixing data types unexpectedly.

```
var pet = "dog";

pet++;

//NaN
```

Undefined

- Found when a value **exists** but isn't defined yet.

```
var firstName;

console.log(firstName);

//undefined


var colors = ["Red", "Yellow", "Blue"];

console.log(colors[3]);

//undefined
```

Null

- Basically the same as undefined, but technically they're different.
- Returned when a value **doesn't exist at all**, usually when dealing with objects in JavaScript (more on that later)

```
null == undefined
//true
null === undefined
//false
```

"Truthy" and "Falsy" values

- **All data in JavaScript, regardless of its type or value, is either "truthy" (true) or "falsy" (false)**
- `NaN`, `undefined`, `false`, `0`, `""` (an empty string) and `null` are all "falsy," and will evaluate as false.
- Every other piece of data in JavaScript is "truthy", and will evaluate as true.
- This mainly matters in "if" statements, but there are other applications...

"Truthy" and "Falsy" values

```
var name = '';

console.log(name);      //undefined

name == false;          //true; undefined is falsy

name === false;         //false; it's undefined
```

"Truthy" and "Falsy" values

- This means that instead of checking whether a variable is defined with an if statement, like this:

```
if(name != undefined) { //Do stuff here }
```

- We can just use the variable name as the condition:

```
if(name) { //Do stuff here }
```

- As long as `name` has a value (that isn't `false` or `0`), it will be "truthy" and the `if` statement will evaluate as "true"

"Truthy" and "Falsy" values

```
var name = document.getElementById('nameInput').value;


if(name){
    console.log("name has a value");
    //as long as name is defined and has a value, it will be truth
} else {
    console.log("name has no value");
    //this code will run if name is undefined or empty
}
```

Once more, all falsy values:

- `NaN`
- `undefined`
- `false`
- `0`
- `""` (an empty string)
- `null`

Every other piece of data is truthy; it exists and has a value.

Important Side Note: Debugging 101

How to solve the puzzle of code gone wrong

How to debug a code issue

- Debugging is important in all code, but especially JavaScript and PHP
- You'll notice things don't go as expected a lot at first
- JavaScript isn't particularly great at explaining its problems
- **The following steps are useful tools not only in programming languages, but in all aspects of coding**
- Any time something goes wrong, ask yourself the following four questions:

1. What did I expect to happen?

- What was the effect I was going for?
- What did I think should have happened?
- What outcome was I anticipating?

- Answering these questions is the first step to understanding where things went wrong.

2. What actually happened?

- Instead of what I was expecting, what occurred instead?
- Did nothing happen? Did the wrong thing happen?
- Did it partially work, but break down somewhere?
- What was the difference between this and what I expected to happen?

- **The difference between expectation and outcome is often the biggest hint at the source of the issue.**

3. What information is needed?

- In order for the code to do what I want it to, what information does it need to have?
- What steps do I need to lay out?
- What data do I need to make sure I include?

- Listing out every piece of information the machine needs to know to fulfill the request or function is often helpful.

4. What information did I provide?

- Did you give the machine all the info it needs?
- Did you “phrase it” in a way the machine understands?
- Is the data in the proper format?
- Is the language and syntax correct?

- Most cases of unexpected outcomes in coding stem directly from failing to provide full, accurate and properly formatted info.

Recap

- Any time anything goes wrong with coding, ask:
    1. **What did I expect to happen?**
    2. **What actually happened (and what's the difference)?**
    3. **What information is needed to make this work?**
    4. **What information did I provide?**
- Repeat these steps for every unsuccessful change that's made

You're Smarter Than You Think

- **Most of the time, the biggest obstacle in debugging is our own pre-conceived idea of how things should be working**
- You'll be amazed at how often saying or writing all of these steps will make the answer clear to you
- **Change your perspective.** Get up. Walk away. Sleep. Talk with somebody else.
- **"I just had to say it out loud"** – the problem is usually us, and often changing perspective is the best answer

Questions?

Ask away