Introduction to Javascript

Professor Larry Heimann
Carnegie Mellon University
Information Systems Program
HTML, CSS, and JS

HTML tells us what an element is

CSS affects what an element looks like

Javascript allows us to manipulate an element
Another view on progressive enhancement

Chapter 1

Progressive Enhancement

Regardless, it is a bottom-up or inside-out building model for a website or application. You first focus on the content and mark it up with semantic and meaningful HTML. This is the first layer, "structure." After the content is properly marked up, we can move onto layer two, "presentation." On the presentation layer, we deal with CSS. The third layer of progressive enhancement, "behavior," we deal with last. This is where we will be spending a lot of time because this is where the JavaScript lives. Figure 1.1 shows the different layers of Web design.

As long as you keep your layers separate, make your site work with only HTML, pretty it up with CSS, and then smooth out the behavior with JavaScript, you will make sure that your content is always accessible.

History

In 2003 in Austin, TX, at South by Southwest, a new term was coined that realigned the Web with the original path of Tim Berners-Lee; this term was announced as progressive enhancement, and ever since, the face of Web design has been changed. The way we build and think moved from focusing on machines (browsers) to a more friendly model, which centered around...
History of Javascript

1995: Netscape creates "LiveScript", renamed to "JavaScript"
1996: Microsoft releases "JScript", a port, for IE3
1997: JavaScript was standardized in the "ECMAScript" spec
2005: "AJAX" was coined, and the web 2.0 age begins
2006: jQuery 1.0 was released
2010: Node.JS was released
2012: ECMAScript Harmony spec nearly finalized
Use of Javascript in web design

Image switchers and slideshows
Lightboxes
Interactive elements (like tabs, sliders & accordions)
Drawing and animation
Track users with cookies, customize experience
Full featured web applications
“In Javascript, there is a beautiful, elegant, highly-expressive language that is buried under a steaming pile of good intentions and blunders.”

Douglas Crawford, Javascript: The Good Parts
Variables

• Use variables to store values.

• Declare, then initialize in 2 statements:

```
var x;
x = 5;
console.log(x);
```

• Or declare and initialize in one statement:

```
var y = 2;
console.log(y);
```

• Re-assign the value later:

```
var x = 5;
x = 1;
```
Data types

**string**: an immutable string of characters:

```javascript
var greeting = 'Hello Kitty';
var restaurant = "Pamela's Place";
```

**number**: whole (6, -102) or floating point (5.8737):

```javascript
var yourAge = 19;
var pi = 3.14159;
```

**boolean**: represents logical values true or false:

```javascript
var dogsAreBest = true;
var catsRule = false;
```

**undefined** vs. **null**: unknown (as of yet) versus explicitly empty:

```javascript
var chancesPiratesWinGame;
var chancesPiratesWinningWorldSeries = null;
```
Variable naming

- Begin with letters, $ or _
- Only contain letters, numbers, $ and _
- Case sensitive
- Avoid reserved words
- Choose clarity and meaning
- Prefer camelCase for multipleWords (instead of under_score)
- Pick a naming convention and stick with it

**OK**

```javascript
var numPeople,
    $mainHeader,
    _num,
    _Num;
```

**Bad**

```javascript
var 2coolForYou,
    soHappy!,
    function,
    [#*@;$!];
```
Expressions

Variables can also store the result of any "expression":

```javascript
var x = 2 + 2;
var y = x * 3;

var name = 'Pamela';
var greeting = 'Hello ' + name;
var title = 'your highness';
var formalGreeting = greeting + ', ' + title;
```
Loose typing

• JS figures out the type based on value, and the type can change:
  ```javascript
  var x;
  x = 2;
  x = 'Hi';
  ```

• A variable can only be of one type:
  ```javascript
  var y = 2 + ' cats';
  console.log(typeof y);
  ```

• Type changing:
  ```javascript
  var y = 2 + ' cats';
  console.log(typeof y);
  ```
Functions

Functions are re-usable collections of statements

First declare the function:

```javascript
function sayMyName() {
    console.log('Hi Prof. H!');
}
```

Then call it (as many times as you want):

```javascript
sayMyName();
```

*Beware: Circular Dependencies*

```javascript
function chicken() { egg(); }
function egg() { chicken(); }
egg();
```
Return values

The return keyword returns a value to whoever calls the function (and exits the function):

```javascript
function addNumbers(num1, num2) {
    var result = num1 + num2;
    return result;  // Anything after this line won't be executed
}

var sum = addNumbers(5, 2);
```

You can use function calls in expressions:

```javascript
var biggerSum = addNumbers(2, 5) + addNumbers(3, 2);
```

You can even call functions inside function calls:

```javascript
var hugeSum = addNumbers(addNumbers(5, 2),
                        addNumbers(3, 7));
```
In javascript, use ‘if’ to tell which statements to execute, based on a condition.

```javascript
if (condition) {
  // statements to execute
}
```

```javascript
var x = 5;
if (x > 0) {
  console.log('x is a positive number!');
}
```

Use ‘else’ to give javascript an alternative statement to execute.

```javascript
var age = 28;
if (age > 16) {
  console.log('Alright, you can drive... if you have a license.');
} else {
  console.log('You have ' + (16 - age) + ' years until you can drive.');
}
```
Control structures

You can use ‘else if’ if you have multiple exclusive conditions to check:

```javascript
var age = 20;
if (age >= 35) {
    console.log('You can vote AND hold any place in government.');
} else if (age >= 25) {
    console.log('You can vote AND run for the Senate.');
} else if (age >= 18) {
    console.log('At least you can vote.');
} else {
    console.log('You have no voice in government...');
}
```
Control structures

The while loop tells JS to repeat statements until a condition is true:

```javascript
while (expression) {
    // statements to repeat
}

var x = 0;
while (x < 5) {
    console.log(x);
    x = x + 1;
}
```

The for loop is another way of repeating statements, more specialized than while:

```javascript
for (initialize; condition; update) {
    // statements to repeat
}

for (var i = 0; i < 5; i = i + 1) {
    console.log(i);
}
```
Javascript Quirks

Demo Time