

# CSS and JS

Pull from upstream!

Slides created in conjunction with: Ilchul Yoon, Nelson Padua Perez

# Agenda

- Office Hours Updates
- Exercise 1 Questions
- Quick Warm Up
- Intro to JS
- JS Logistics

# Office Hours Room Change

Moved to AVW 4122!

8am to 8pm

My office is the same: IRB 2224

# Exercise Questions?

Remember to Validate!!

# Warm Up

Write an example html page where:

`.names p {...}`

will style a paragraph, but

`.names > p{...}` **will not.**

# JavaScript

- Finally some programming!
- JavaScript is a programming language that allows us to:
  - Create interactive web pages
  - Control a browser application
    - Open and create browser windows
    - Download and display contents
  - Interact with the user
  - Interact with HTML Forms

# JS and ECMAScript

- JavaScript implements ECMAScript

## What is ECMAScript?

- A scripting language standard
- ActionScript and JScript are other implementations

# How is JavaScript different?

JavaScript implementation includes:

- ECMAScript
- DOM (Document Object Model)
- BOM (Browser Object Model)



# Notable Changes To JavaScript

- ECMAScript 5
  - Added “use strict”
  - Added more JSON support
  - Added Array iteration methods
- ECMAScript 2015
  - Added let and const
  - Added default parameter values
  - More Array Methods
- More recently: ECMAScript 2018
  - Added Asynchronous iteration
  - Added rest/spread properties

# JavaScript Engine

Example: templateJS.html

- JavaScript engine process JavaScript code
  - Safari: JavaScriptCore
  - Chrome: V8
  - Firefox: Spidermonkey
  - Edge: Chakra
- To write JavaScript programs you need a web browser and a text editor
- A JavaScript program can appear:
  - In a file by itself typically named with the extension .js
  - In html files between a <script> and </script> tags.

# Processing HTML with JS

- DOM – Document Object Model

- Structured representation of the HTML page
- Every HTML element is represented as a node
- Browser uses HTML to build the DOM and can fix problems with the HTML so a valid DOM is generated

- Lifecycle

- Set the user interface
  - Parse the HTML and build the DOM
  - Process (execute) JavaScript code
- Enter a loop and wait for events to take place

# Processing HTML with JS

- When JavaScript is seen in a page, the DOM construction is halted and JavaScript code execution is started.
- JS can modify the DOM (e.g., creating, modifying nodes)
  - One reason why `<script></script>` elements appear at the bottom of a page (speed)

# Event Handling

- Relies on a single-threaded execution model
- An event queue keeps track of events that have taken place, but have not been processed (event-handler function for the event has not been called)
- All generated events (whether are user-generated or not) are placed in the event queue in the order they were detected by the browser
  - The browser mechanism that detects events and that adds them to the event queue is separate from the thread that is handling the events

# Browser's Global Objects

- Browsers provide two global objects: window and document
- window object – represents the window in which a page resides
  - Provides access to other global objects (e.g., document)
  - Keeps track of user's global variables
  - Allows JavaScript to access Browser's APIs
- document object
  - Property of the window object that represents the DOM of the current page
  - Via this object you can access & modify the DOM

# Types of JavaScript Code

- **Function Code**
  - Code contained in a function
- **Global Code**
  - Code placed outside all functions
  - Automatically executed by JS engine
- As in Java, a stack is used to keep track of function calls. Each function call generates a function execution context (stack frame)
- There is one frame called the global execution context created when the JS program starts executing.

# How do we run JavaScript?

- Chrome (or any browser)
  - Right click -> inspect
- Node
  - Make sure you have it installed
- Within HTML



Let's take a look at an example!

## Open TemplateJS

# JavaScript Comments

- Comments in JavaScript

- Used to provide information to the programmer
- Used to identify sections in your code
- Ignored by the JavaScript interpreter

- Two types of comments

- Inline comment `// This is a comment until the end of the line`
- Block comment `/* The following is a  
  
comment that spans  
several lines */`

# JavaScript Data Types

- JavaScript has no class concept (at least until ES6)
- Two kinds of types:
  - Primitive types – simple data stored as is
  - Reference types – references to locations in memory
- Primitive data types in JavaScript
  - Null – has value null
  - Boolean – values true or false
  - Number – numeric value
  - String – character sequence delimited by single or double quotes
  - Undefined – has as value undefined (values associated with variables that are not initialized)

# typeof

## typeof operator

- Returns string indicating the type of data
- Note: typeof null will returns “object”

# JavaScript Data Types cont.

- Reference types represents objects in JavaScript
- Reference values are instances of reference types and considered objects
- Object – collection of properties
  - Property – string that is associated with a value
  - Value – could be a primitive, object, function

# Object Creation

- Object creation

*We'll talk about this more later!*

```
var myFirstObject = new Object();  
var mySecondObject = {  
  id: 789,  
  name: "Rose Smith"  
}; // object literal
```

- JavaScript relies on garbage collection

- When an object is no longer needed set the variable to null

# Type Conversion

- Most of the time implicit transformations will take care of transforming a value to the expected one
- Example:
  - `var age = 10;`  
`var s = "John Age: " + age;`
- Mechanism to transform values:
  - Converting number to string
    - `var stringValue = String(number);`
  - Converting string to number
    - `var number = Number(stringValue);`
    - `var number = parseInt(stringValue);`
    - `var number = parseFloat(stringValue)`

# var vs let vs const

var: how variables were created pre-ECMA2015

- Does not have block scope
- Kind of a loose cannon

let: defines a ***binding***

- Can be accessed outside a block
- More “strict”

const: Creates a constant that cannot be modified after it's initial assignment



# Comparisons

- You can compare values by using the following operators
  - `===` Return true if the values are equal, false otherwise (e.g., `x === y`)
  - `!==` Returns true if the values are different, false otherwise (e.g., `x !== y`)
- `==` and `!=` Not as strict as previous equality operators
- Relational Operators
  - `<` Less than
  - `>` Greater than
  - `<=` Less than or equal
  - `>=` Greater than or equal

# Dialog Boxes- Our go-to user input

- We can perform input and output via dialog boxes
- Input via ***prompt***
  - returns a string

If you need to perform some mathematical computation you might need to explicitly convert the value read into a number

Example: [inputOutput.html/network.html](#)

# Control Structures

All are the same as Java:

- While, do while, for loops
- If statements
  - nested/cascading
- Switch
  - break

Example: `sqrTable.html`

# What is “use strict”?

- JavaScript's strict mode, introduced in ES5
- A way to opt in to a restricted variant of JavaScript, thereby implicitly opting-out of "sloppy mode".
- Several changes to normal JavaScript semantics:
  - Makes JavaScript silent errors throw errors
  - Prohibits some syntax likely to be defined in future versions of ECMAScript.
- Examples not allowed
  - Declaring function in blocks `if (a < b) { function f() {} }`
  - Setting a value to an undeclared variable

# Strict Mode

- Allows for error checking both globally or within a function
- Use the strict mode pragma
  - “use strict”;
- If pragma used outside of a function, it applies to all the script
- It can appear in a function or out
- Forces variables to be declared first
- Cannot use reserved words

# Console object

- Allow us to view JavaScript errors and user messages
- console object functions
  - log : General message
  - info : Informational message
  - error : Error message
  - warn : Warning message

Let's test them in the browser!

Example: consoleEx.html

# Built-in Types

- Object – generic object
- Array – list of values (numerically indexed)
- Function
- Error – runtime error
- Date – date / time
- RegExp – regular expression
- Many of built-in type have a literal form that enables you to define a value without explicitly creating an object (using new)
- The typical function definition is based on a literal form

# Primitive Wrapper Types

- JavaScript promptly coerces between primitives and objects when a property of the type is accessed.
- Three wrapper types: **Boolean**, **String**, and **Number**
- Primitive wrapper types simplify working with primitives
- Wrapper types are automatically created when needed

Example: [wrapper.html/wrapperType.html](http://wrapper.html/wrapperType.html)



# Global Objects

- ECMAScript defines a global object
- All functions and variables defined globally become part of the global object
- Some functions that are part of the Global object
  - isNaN()
  - parseFloat()
  - parseInt()
  - eval() : evaluates JS code represented as a string
  - isFinite()
  - decodeURI()

# Global Object

- Some properties that are part of the Global object
  - NaN
  - undefined
  - Object : Constructor for Object
  - Array : Constructor for Array
  - Function : Constructor for Function
  - Number : Constructor for Number
  - String : Construct or for String
  - Date : Construct
  - Error : Constructor
  - RegExp : Constructor
- ECMAScript also defines the Math object

# WTWAW

After today make sure you know:

- What is JavaScript?
- How does JS fit in with the DOM?
- What is the JavaScript LifeCycle?
- How does JS handle events?