# Internet Programming
# Week 2

Instructor:
Daniel Slack, P. Eng.
Applied Computer Science
University of Winnipeg

# Chapter 2, 3 and part of 13

Program Structure

Functions and Objects

# JavaScript

- HTML is known as the page structure
- CSS is known as the page presentation
- A great looking page can be made with these two technologies
- But JavaScript is required to add behaviour to these pages
- JavaScript is used to create an interactive experience

# JavaScript

- Goal
  - Write JavaScript that runs in the browser when page is loaded
  - Code that
    - Responds to user actions
    - Updates or changes the page
    - Communicates with Web Services
    - Interacts with Sensors
    - Manages Offline State

# JavaScript

- Writing
  - Can use inline <script> tags
    - Like CSS, not maintainable, reusable
  - Should be in separate files
    - E.g. index.html could load index.js, app.js, etc.
      - Maintainable by other teams
      - Better organization
        - todo.list.js, todo.add.js, todo.edit.js

# JavaScript

- Loading
  - The browser retrieves and loads your page
    - `<script src='your.code.js'></script>`
  - Parsing content from top to bottom
  - JavaScript:
    - Browser parses code
    - Checks for correctness
    - Executes code
    - Browser also builds the DOM

# JavaScript

- Running
  - JavaScript executes continuously
  - DOM is used to
    - Examine the page
    - Change it
    - Receive events from it
    - Ask the browser to retrieve other data from the server

# JavaScript

- A script element is all that is necessary to start coding

```
<html>
…
<script>
        var myVariable = 123;
</script>
…
</html>
```

# Make a Statement

- Create a variable and assign values
- Perform addition
- Perform calculations
- Use built-in JavaScript Libraries

```
var temp = 98.6;
var beanCounter = 4;
var reallyCool = true;
var motto = “I Rule”;
temp = (temp - 32) * 5 / 9;
motto = motto + “ and so do you!”;
var pos = Math.random();
```

# Loops

- While and Do-While loops

```
while (beanCounter > 0)
{
  processBeans();
  beanCounter = beanCounter – 1;
}
```

# Conditional Statements

- Conditionals (if/elseif/else)

```
if (isReallyCool)
  invite = “You’re invited!”;
else if (!isReallyCool && isKindaCool)
  invite = “Maybe next week”;
else
  invite = “Sorry, we’re at capacity.”;
```

# Declaring a Variable

- Declare your variable

```
var scoops;
```

  - Notice a type is not required
  - The above statement just creates a generic container
    - Can hold anything at this point

```
scoops = 10;
Scoops = “More”;
```

# Declaring a Variable

- Integers:
  - `var winners = 2;`
- Floating point numbers:
  - `var boilingPt = 212.0;`
- Strings:
  - `var name = “Dr. Evil”;`
- Boolean:
  - `var isEligible = false;`

# Declaring a Variable

- Next, a value can be specified in a few ways
  - Value can be a literal (number of string)

```
var scoops = 10;
```

  - Value can be the result of an expression

```
var scoops = totalScoops / people;
```

  - Or use one of JavaScript's internal library functions

```
var scoops = Math.random() * 10;
```

# Declaring a Variable

- What is the value of a variable when nothing is assigned?
  - The variable will be assigned the value *undefined*
  - This is another JavaScript value and type
- Does JavaScript not have types?
  - JavaScript uses dynamic typing
  - Users don't have to specify a type
  - JavaScript interpreter will figure out what type to use
    - During execution

# HTML vs JavaScript

- HTML is made of declarative markup
  - Describes a set of nested elements that make up your page
- JavaScript is meant for describing computations
- They have something in common
  - The DOM
- DOM allows JavaScript to communicate with your page
  - And *vice versa*

# Introduction to the DOM

- The browser adds an object to the DOM for each element in your page
- Scenario: You want to change the property of an object in the DOM
- Solution requires a reference to an object (*i*.e. an element)
- The reference can be retrieved using the element's id
- Recall: the id attribute specifies an unique id for an HTML element

# Introduction of the DOM

- *document* object represents the entire page
  - Contains the complete DOM
  - We can ask it to find an element with a specific id

```
document.getElementById("elemID");
```

- Example Time!

# DOM

- DOM functionality:
  - Get elements from the DOM
    - Using ids
    - Using tag names
    - Using (HTML) class names
    - Using Attributes
    - Can retrieve sets of elements from the page
    - Can get text from form input elements
    - Query using selectors

# DOM

- Create or Add elements
  - Can create new elements
  - Can add these new elements to the DOM
  - Any changes are immediately rendered by the browser
- Remove elements from the DOM
  - Get a reference to a parent element
  - Then, remove any of its children
  - Browsers immediately updates
- Get and Set attributes of elements
  - We've just changed text
  - Can do the same with attributes

# Aside: Arrays

- Create a new array

```
var tempByHour = new Array();
var tempByHour = [];
```

- Get the size of the array

```
var numItems = tempByHour.length;
```

- Everything else is the same as in Java

# Examples

- Let us look at some examples of these concepts

# Functions

- We have briefly seen function definitions in previous lectures
- A function takes the form of

```
function functionName (zero, or, more, parameters) {
…
//Optionally return a value
}
```

# Functions

- How are arguments passed to functions in JavaScript?
  - By value
    - A primitive value is copied into the function parameter
- Is it possible to change values in a function?
  - Only global variables can be changed in a function

# Functions

- Is ‘var’ necessary when defining objects?
  - Any new variable missing the ‘var’ keyword is created as global variable
    - Even if within a function
    - Be careful

# Functions

- Functions are also values
- Variables are used to store primitives, arrays, *etc.*
- Also possible to assign a function to a variable
  - Since it is a value

```
var addOne = function(num) {
        return num + 1;
}
var plusOne = addOne;
var result = plusOne(1);
```

  - Where have we seen this before?

# Functions

- Knowing functions are values allows you to do some useful things
  - You can store values in variables or arrays
  - You can pass them as arguments to functions
  - You can assign them to the properties of objects

# Functions

- Can also give a function a name

```
function increaseByOne(num) {
        return num + 1;
}

var result = increaseByOne(1);
```

# Functions

- With those declarations, we can do:

```
function myOnload() {
        console.log('This has loaded');
}
window.onload = myOnload;
Or even shorter:
window.onload = function() {
        console.log('This has loaded');
}
```

- What about the other way of defining?

# Functions

- When a variable is declared, JavaScript 'hoists' it to the top of the 'block'
- If we had:

```
console.log('Starting: '+myName);    //Outputs undefined
var myName = 'Dan';
console.log('Ending: '+myName);      //outputs Dan
```

- Is interpreted as:

```
var myName;
console.log('Starting: '+myName);    //Outputs undefined
myName = 'Dan';
console.log('Ending: '+myName);
```

# Functions

- BUT!!!
  - Named functions are treated a bit differently
  - The entire function is 'hoisted', not just the variable
  - This is why we can do:

```
window.onload = myOnload;
function myOnload() {
        console.log('This has loaded');
}
```

# Functions

- This means that we CANNOT do:

```
window.onload = myOnload;
var myOnload = function () {
        console.log('This has loaded');
}
```

- But, we could do:

```
var myOnload = function () {
        console.log('This has loaded');
}
window.onload = myOnload;
```

# Functions

- JavaScript has a specific concept of “scope”
  - Global vs. Local
  - Where we declare a variable is important
  - Declared outside of a function, variables are global
  - Declared inside of a function, local to that function

```
var myName = “Dan”;
function writeOutName(name) {
        var myName = name;
        console.log(myName);  //outputs Daniel
}
writeOutName(“Daniel”);
console.log(myName);          //outputs Dan
```

# Functions

- Closures
  - Declare a function inside another, and use local variable?

```
function outputPrefixName() {
        var localName = “Dan”;
        return function(prefix) {
                return prefix+” “+localName;
        }
}
var prefixFn = outputPrefixName();
console.log(prefixFn(‘Mr.’));                  //Mr. Dan
console.log(prefixFn(‘A person named’));    //A person named Dan
```

- localName is available to inner function
- JS detects its use and “closes around” to make it available

# Functions

- Lets try some examples out…

# Objects

- Start with an example
- e.g. a dog
  - A dog could have the following properties
    - Name (string)
    - Weight (integer)
    - Breed (string)
    - List of activities they enjoy (Array of strings)

# Objects

- Example of creating an object

```
var fido = {
        name: “Fido”,
        weight: 40,
        breed: “Mixed”,
        loves: [“walks”, “fetching balls”]
};
```

- Properties are accessed with dot notation

```
if (fido.weight > 25)
        alert(“WOOF”);
else
        alert(“YIP”);
```

# Objects

- Properties can also be accessed using a string with [] notation

```
var breed = fido[“breed”];
if (breed = “mixed”) {
        alert(“Best in show”);
}
```

- Enumerate all an object’s properties

```
var prop;
for (prop in fido) {
        alert(“Fido has a “ + prop + “ property “);
}
```

# Objects

- Properties can be added or deleted at any time
  - Add a property

```
fido.age = 5;
```

  - Delete a property

```
delete fido.age;
```

# Passing Objects to Functions

- Arguments are passed by value
  - e.g. passing an integer results in the function receiving a copy
    - Thus, original variable cannot be modified by the function
- Same is true for objects
  - Sort of!

# Passing Objects to Functions

- Variables assigned objects only contain a reference to the object
- Thus, function call copies reference
  - Any change to object property is made to original object

# Object Behaviour

- Example of adding a method to an object

```
var fido = {
   name: “Fido”,
   weight: 40,
   breed: “Mixed”,
   loves: [“walks”, “fetching balls”];
   bark: function () {
        alert(“Woof woof!”);
   }
};
```

# Defining Objects

- Was the dog object defined thus far useful?

# Object Constructor

```
function Dog(name, breed, weight) {
   this.name = name;
   this.breed = breed;
   this.weight = weight;
   this.bark = function() {
        if (this.weight > 25) {
              alert(this.name + “ says Woof!”);
        } else {
              alert(this.name + “ says Yip!”);
        }
   };
}
```

# Constructor Observations

- By convention, the constructor names are capitalized
- Property names and parameter names do not have to be the same
  - Usually are by convention
- Notice the syntax differs from object syntax

# Using the Constructor

```
var fido = new Dog(“Fido”, “Mixed”, 38);
var tiny = new Dog(“Tiny”, “Chawalla”, 8);
var clifford = new Dog(“Clifford”, “Bloodhound”, 65);

fido.bark();
tiny.bark();
clifford.bark();
```