Events and the DOM

Responding to User Interaction

Recap

Last week was a flurry of the basic javascript language constructs including...

- Primitive types
- Variables
- Arrays
- Conditional Statements
- Looping
- Functions
- Objects

Variables - Recap

Assigning variables to names with the = sign.

```
var company = “Main Street Host”;

var favoriteColor = “Blue”;

var shouldWatchBetterCallSaul = true;

var metsWorldSeries = 2;
```

Arrays - Recap

Arrays can be thought of as a collection or variables or values. They can contain any combination of types, objects or even functions.

```
var bestRestaurants = [“Toutant”, “Shango”, “Bar Bill”];

var randomListOfThings = [“Sidewalk”, 4, false, “Captain America”];

console.log(randomListOfThings[2]); // false
```

Looping - Recap

When you need to execute code repeatedly or navigate through an array, use a *for-loop*.

```
var furniture = [“Chair”, “Table”, “Desk”];

for (var i = 0; i < furniture.length; i++) {

    console.log(furniture[i]);

}
```

Functions and If-Statements - Recap

Functions are used to share code, encapsulate data or make your code more readable.

```
function canIBuyACar(carCost) {

    var myMoneyTotal = 100;

    if (carCost > 100) {

        return false;

    }

    return true;

}
```

Objects - Recap

Objects are special types in javascript used to group relevant data together. Use objects when modelling a real world system.

```
var dog = {

    name: ‘Marmaduke’,

    speak: function () {

        alert(“Woof”);

    }

};
```

Outline

- Any Questions before moving on?
- Events
- DOM

Events

Events are javascripts way of allowing your code to respond to outside activity or user interaction.

When a user clicks a button, then we can say a button has been *clicked* and a *click event* will be *raised*.

For example:

- If a web page is done loading, then the browser will *raise* a *loaded event*.
- A user selecting a different option in a select box will *raise* a *changed event*.

Events

HTML events are being raised by (almost) every action the user is taking.

In order to respond to these events, we need to use javascript in one of two ways.

- Attach *event handlers* in HTML
- Use unobtrusive javascript event handlers - for another time

Events - Handlers

The most basic way of responding to events is to add code directly into the HTML markup.

```
<button onclick="console.log('Clicking');">Click Me</button>
```

Whenever this button is pressed, the code inside of the *onclick* attribute is run.

Events - Handlers

A better way to attach event handlers to HTML events is by assigning functions instead of inlining the code.

```
<button onclick=”buttonClicked()”>Click Me</button>

function buttonClicked() {

    console.log(‘Button clicked’);

}
```

Events - Handlers

When the function is called from an event, a special parameter is passed to the function to get more information about the action.

```
<button onclick=”buttonClicked(event)”>Click Me</button>

function buttonClicked(event)  {

    console.log(event);

}
```

Note: the name of the parameter MUST be event, it is passed in as part of a closure.

Events - Types

A few common event types you'll see are:

Action	Event Name
- Clicking any object (buttons, links etc...)	- click
- Changing the value in a Select list	- change
- Dragging an item across the screen	- dragstart
- Typing into a textbox	- keydown , keyup

Events - Types

Use keydown or keyup to respond to a user typing into a textbox.

```
<input type=”text” placeholder=”Start Typing” onkeyup=”
userIsTyping(event)” />

function userIsTyping(event) {

    console.log(event.target.value);

    // We’ll describe event.target later on

}
```

DOM - Document Object Model

DOM stands for Document Object Model.

The DOM is javascript's *object* representation of the HTML.

It is what allows us to modify the webpage with javascript scripting code.

The DOM helps us when we need to update the webpage based on user input or server response.

DOM - Selection

Before we can manipulate items on a webpage, we need to be able to assign a reference to that object into a *variable*.

We can search the page and assign an object to a variable using the built-in function...

```
var myButton = document.getElementById(“myFirstButton”);
```

```
<button id=”myFirstButton”>Click Me</button>
```

DOM - Selection

`document.getElementById(“idOfThing”);` works by searching the web page for an HTML element with the id “idOfThing”.

Here, “idOfThing” is a parameter meant to represent the ID of an HTML element.

If there is more than one element on the page with the same id attribute, the function will return the first value on the page.

This is a main reason why the ID attribute of an HTML element should be unique.

DOM - Selection

Once we have a reference to a DOM variable, we can inspect the object for certain built-in properties.

```
<input type=”text” id=”firstName” value=”Starting Value” />

var textBox = document.getElementById(‘firstName’);

console.log(textBox.value);
```

DOM - Selection

You can select multiple elements into an array by using the function...

```
var listItems = document.getElementsByClassName(“list-item”);
```

```
<ul>
    <li class=”list-item”>First Item</li>
    <li class=”list-item”>Second Item</li>
</ul>
```

DOM - Selection

`document.getElementsByClassName("list-item")` - this is an effective way of getting multiple elements, but it could potentially be slow and is not supported by all browsers.

Cross-browser predictability and support is the main reason frameworks like jQuery and now Angular are popular.

You most likely will not see `document.getElementsByClassName("list-item")` used in the wild except in pure-vanilla js circumstances.

DOM - Manipulation

In general, we will attach event handlers to HTML in order to provide feedback on certain input or update the screen based on a response.

Fortunately, updating the screen is as simple as reassigning some properties on the DOM object.

```
var textBox = document.getElementById(“fullName”);

textBox.value = “My Full Name”;
```

DOM - Manipulation

You can do more than just update the value of a textbox. Updated the CSS properties is just as simple.

```
<div id=”myFirstDiv”>This is a div with some text.</div>

var myFirstDiv = document.getElementById(“myFirstDiv”);

myFirstDiv.style.color = “red”;
```

DOM - Manipulation

In addition to colors you can also update:

- Size
- Location
- Visibility
- Opaqueness

Any CSS property can be added or modified using javascript.

DOM - Combining

Now that we know how to select and manipulate elements, we can combine utilities to create fully-functional user interfaces.

```
<input type=”text” placeholder=”First Name” onkeyup=”onKeyUp(event)” />

<div id=”validation”>Is This Valid?</div>

function onKeyUp(event) {

    if (event.target.value === “invalid”) {

        document.getElementById(“validation”).style.color === “red”;

    }
}
```

DOM - Combining

One of the most basic uses for javascript is validating form input to make sure we don’ t collect bad or malformed data.

- In the following group example, make sure that the all fields in the form are filled out.
- Next, show and hide fields based on the selection of an input.

Group Exercises

- Fill in the blanks. There are a few fill-in the blank exercises at https://github.com/kylepace/JsWorkshop2016/tree/master/week2.
- Create three buttons and a separate paragraph section. Label the buttons Blue, Red and Green. When clicking the button, change the paragraph color to the label of the button clicked.
- Move the image on the screen.

External Learning Material

- https://developer.mozilla.org/en-US/docs/Web/Events
- http://www.w3schools.com/js/js_htmldom.asp