JAVASCRIPT & JQUERY

REMINDER!

- What is allowed to be used in your project submission:
  - HTML 5
  - CSS 3
  - Javascript (and jQuery)
  - Flask
  - Any extensions that we have used in class
  - Any code that ***you*** wrote

- *Nothing* else is allowed
- *Plagiarism will be dealt with severely*

JAVASCRIPT PSEUDO-CLASSES

Object Oriented Design

Without Classes

- JavaScript is not a full-fledged object-oriented programming language.
- JavaScript has no formal class mechanism.
- Many common features of object-oriented programming, such as inheritance and even simple methods, must be arrived at through non-intuitive means.
- These “strange” techniques give JavaScript a bad reputation, but can easily be mastered.

JavaScript Objects-Recall

Not full-fledged O.O.

Objects can have **constructors**, **properties**, and **methods** associated with them.

There are objects that are included in the JavaScript language; you can also define your own kind of objects.

Constructors-Recall

Normally to create a new object we use the new keyword, the class name, and ( ) brackets with *n* optional parameters inside, comma delimited as follows:

```
var someObject = new ObjectName(p1,p2,..., pn);
```

For some classes, shortcut constructors are defined

```
var greeting = "Good Morning";
```

vs the formal:

```
var greeting = new String("Good Morning");
```

Properties-Recall

Use the dot

Each object might have properties that can be accessed, depending on its definition.

When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property.

```
//show someObject.property to the user
alert(someObject.property);
```

Methods-Recall

Use the dot, with brackets

Objects can also have methods, which are **functions** associated with an instance of an object. These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.

```
someObject.doSomething();
```

Methods may produce different output depending on the object they are associated with because *they can utilize the internal properties of the object.*

Objects Included in JavaScript-Recall

A number of useful objects are included with JavaScript including:

- Array
- Boolean
- Date
- Math
- String
- DOM objects

User Defined Object-Recall

```
<script>
var person = {
firstName: "John",
lastName : "Doe",
id       : 5566,
fullName : function() {
    return this.firstName + " " + this.lastName;
    }
};
</script>
```

Using Object Literals

Without Classes

An object can be instantiated using object literals.

Object literals are also known as **Plain Objects**.

Object literals are a list of key-value pairs with colons between the key and value with commas separating key-value pairs.

Using Object Literals

Consider a die (single dice)

```
//define a 6 faced dice, with a red color.
var oneDie = { color : "FF0000",
               faces : [1,2,3,4,5,6]
             };
```

These elements can be accessed using dot notation.

For instance, one could change the color to blue by writing:

```
oneDie.color="0000FF";
```

Emulate Classes with functions

We told you this would get weird

It is possible to mimic many features of classes by using functions to encapsulate variables and methods together.

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}
```

LISTING 15.1 Very simple Die pseudo-class definition as a function

Emulate Classes with functions

We told you this would get weird

It is possible to mimic many features of classes by using functions to encapsulate variables and methods together.

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}
```

LISTING 15.1 Very simple Die pseudo-class definition as a function

Instantiation looks like:

```
var oneDie = new Die("0000FF");
```

Emulate Classes with functions

Add methods to the object – mimic methods

One technique for adding a method inside of a class definition is by assigning an anonymous function to a variable

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];

   // define method randomRoll as an anonymous function
   this.randomRoll = function() {
      var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
      return faces[randNum-1];
   };
}
```

LISTING 15.2 Die pseudo-class with an internally defined method

Emulate Classes with functions

Add methods to the object – mimic methods

One technique for adding a method inside of a class definition is by assigning an anonymous function to a variable

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];

   // define method randomRoll as an anonymous function
   this.randomRoll = function() {
      var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
      return faces[randNum-1];
   };
}
```

```
var oneDie = new Die("0000FF");
console.log(oneDie.randomRoll() + " was rolled");
```

Emulate Classes with functions

Not very efficient

Defining methods as we have shown is not a memory-efficient approach because each inline method is redefined for each new object!

x : Die

```
this.col = "#ff0000";
this.faces = [1,2,3,4,5,6];

this.randomRoll = function(){

      var randNum = Math.floor
      ( (Math.random() *
      this.faces.length) + 1);

      return faces[randNum-1];

};
```

y : Die

```
this.col = "#0000ff";
this.faces = [1,2,3,4,5,6];

this.randomRoll = function(){

      var randNum = Math.floor
      ( (Math.random() *
      this.faces.length) + 1);

      return faces[randNum-1];

};
```

Using Prototypes

Prototypes

A better approach is to define the method just once using a *prototype* of the class.

- **Prototypes** are used to make JavaScript behave more like an object-oriented language.
- The prototype properties and methods are defined *once* for all instances of an *object*.
- Every object has a prototype

Using Prototypes

moving the randomRoll() function into the **prototype**.

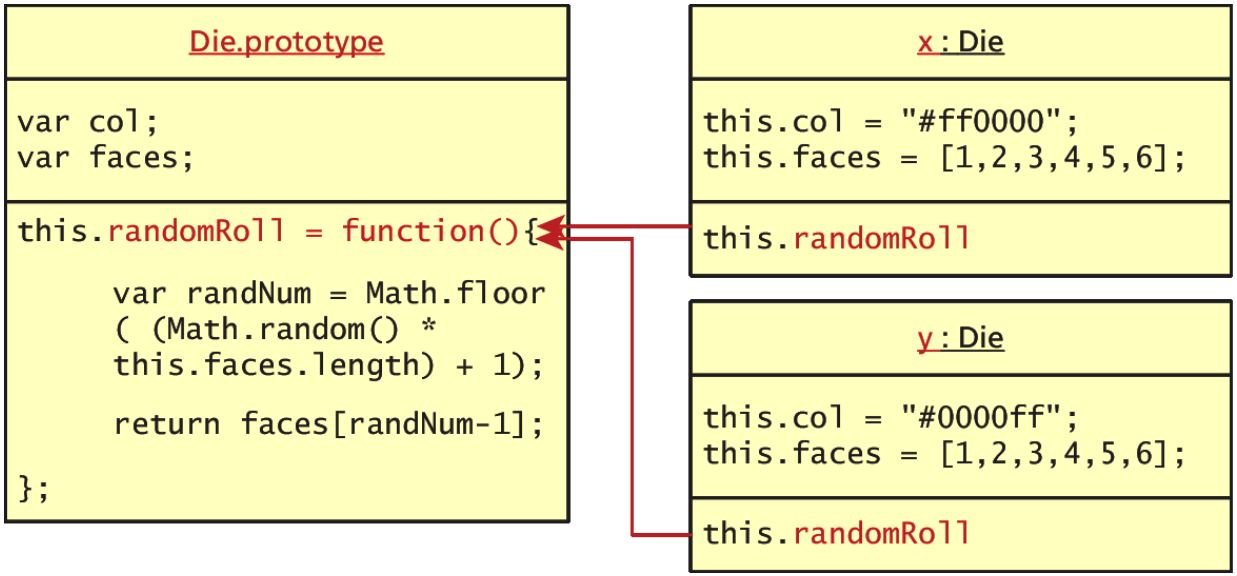
```
// Start Die Class
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}

Die.prototype.randomRoll = function() {
   var randNum = Math.floor((Math.random() * this.faces.length) + 1);
   return faces[randNum-1];
};
// End Die Class
```

Using Prototypes

No duplication of methods

Since all instances of a Die share the same prototype object, the function declaration only happens one time and is shared with all Die instances.



More about Prototypes

There's more?

Every object (and method) in JavaScript has a prototype.

> *A prototype is an object from which other objects inherit.*

The above definition sounds almost like a class in an object-oriented language, except that a prototype is itself an *object,* not an abstraction

More about Prototypes

Extend any Object

In addition to using prototypes for our own pseudo-classes, prototypes enable you to *extend* existing classes by adding to their prototypes!

Below we extend String:

```
String.prototype.countChars = function (c) {
   var count=0;
   for (var i=0;i<this.length;i++) {
      if (this.charAt(i) == c)
         count++;
   }
   return count;
}
```

More about Prototypes

Extending String, for example

Now any **new** instances of String will have this method available to them while existing strings will not.

Now we can use the new method in all new Strings.

The following example will output

Hello World has 3 letter l's.

```
var greeting = new String("Hello World”);


console.log("greeting has" +
greeting.countChars("l") + " letter l's");
```

JQUERY FOUNDATIONS

jQuery
Framework

A **library** or **framework** is software that you can utilize in your own software, which provides some common implementations of standard ideas.

Many developers find that once they start using a framework like jQuery, there's no going back to "pure" JavaScript because the framework offers so many useful shortcuts and succinct ways of doing things

https://jquery.com

jQuery

A 1 slide history

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities
- …

Including jQuery

Let's get started

You must either:

- link to a locally hosted version of the library
  - Download the jQuery library from: https://code.jquery.com
  - From Moodle – Downloads section
- Include jQuery from a CDN
  - Use an approved third-party host, such as Google, Microsoft, or jQuery itself

Including jQuery

```
<head>
<script src="jquery-3.4.1.min.js"></script>
</head>
```

```
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/
jquery/3.4.1/jquery.min.js"></script>
</head>
```

```
<head>
<script
src=https://ajax.aspnetcdn.com/ajax/jQuery/j
query-3.4.1.min.js async></script>
</head>
```

jQuery Selectors

Should ring a bell

When discussing basic JavaScript we introduced the **getElementByID()** and **querySelector()** selector functions in JavaScript.

```
var x = document.getElementById("main");
var el = document.querySelector(".myclass");
```

Although the advanced **querySelector()** methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers

jQuery introduces its own way to select an element, which under the hood supports a myriad of older browsers for you!

jQuery Selectors

The easiest way to select an element yet

The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework,

jQuery()

Is reserved for selecting elements from the DOM.

Because it is used so frequently, it has a shortcut notation and can be written as

$()

Basic Selectors

All the way back to CSS

The four basic selectors are:

- $("*") **Universal selector** matches all elements (and is slow).
- $("tag") **Element selector** matches all elements with the given element name.
- $(".class") **Class selector** matches all elements with the given CSS class.
- $("#id") **Id selector** matches all elements with a given HTML id attribute.

https://learn.jquery.com/using-jquery-core/selecting-elements/

Basic Selectors

All the way back to CSS

For example, to select the single element with id="grab" you would write:

```
var singleElement = $("#grab");
```

To get a set of all the <a> elements the selector would be:

```
var allAs = $("a");
```

These selectors replace the use of getElementById() entirely.

More CSS Selectors

jQuery's selectors are powerful indeed

In addition to these basic selectors, you can use the other CSS selectors:

- attribute selectors,
- pseudo-element selectors, and
- contextual selectors

Selection examples

```
<a href="index.html"></a>                          <!-- 1 -->
<a id="second-link"></a>                           <!-- 2 -->
<a class="example"></a>                            <!-- 3 -->
<a class="example" href="about.html"></a>          <!-- 4 -->
<span class="example"></span>                      <!-- 5 -->
```

`$("a")` → selects 1, 2, 3, and 4

`$(".example")` → selects 3, 4, and 5

`$("#second-link")` → selects 2

`$("[href = 'index.html']")` → selects **only** 1

`$("a:eq(1)")` → selects 2

`$("a:not(.example)")` → selects 1 and 2

Attribute Selector

Really a review of CSS

Recall from CSS that you can select

- by attribute with square brackets
  - [attribute]
- Specify a value with an equals sign
  - [attribute=value]
- Search for a particular value in the beginning, end, or anywhere inside a string
  - [attribute^=value]
  - [attribute$=value]
  - [attribute*=value]

Attribute Selector

Really a review of CSS

Consider a selector to grab all <img> elements with an src attribute beginning with */artist/* as:

```
var artistImages = $("img[src^='/artist/']");
```

Pseudo-Element Selector

Not to be confused with the pseudo-classes in JavaScript

pseudo-element selectors are also from CSS and allow you to append to any selector using the colon and one of

- :link
- :visited
- :focus
- :hover
- :active
- :checked
- :first-child, :first-line, and :first-letter

Pseudo-Element Selector

Not to be confused with the pseudo-classes in JavaScript

Selecting all links that have been visited, for example, would be specified with:

```
var visitedLinks = $("a:visited");
```

Contextual Selector

Put it into context

Contextual selectors are also from CSS. Recall that these include:

- descendant (space)
- child (>)
- adjacent sibling (+)
- and general sibling (~).

To select all <p> elements inside of elements you would write

```
var para = $("div p");
```

Content Filters

Above and Beyond CSS

The **content filter** is the only jQuery selector that allows you to append filters to all of the selectors you've used thus far and match a particular pattern.

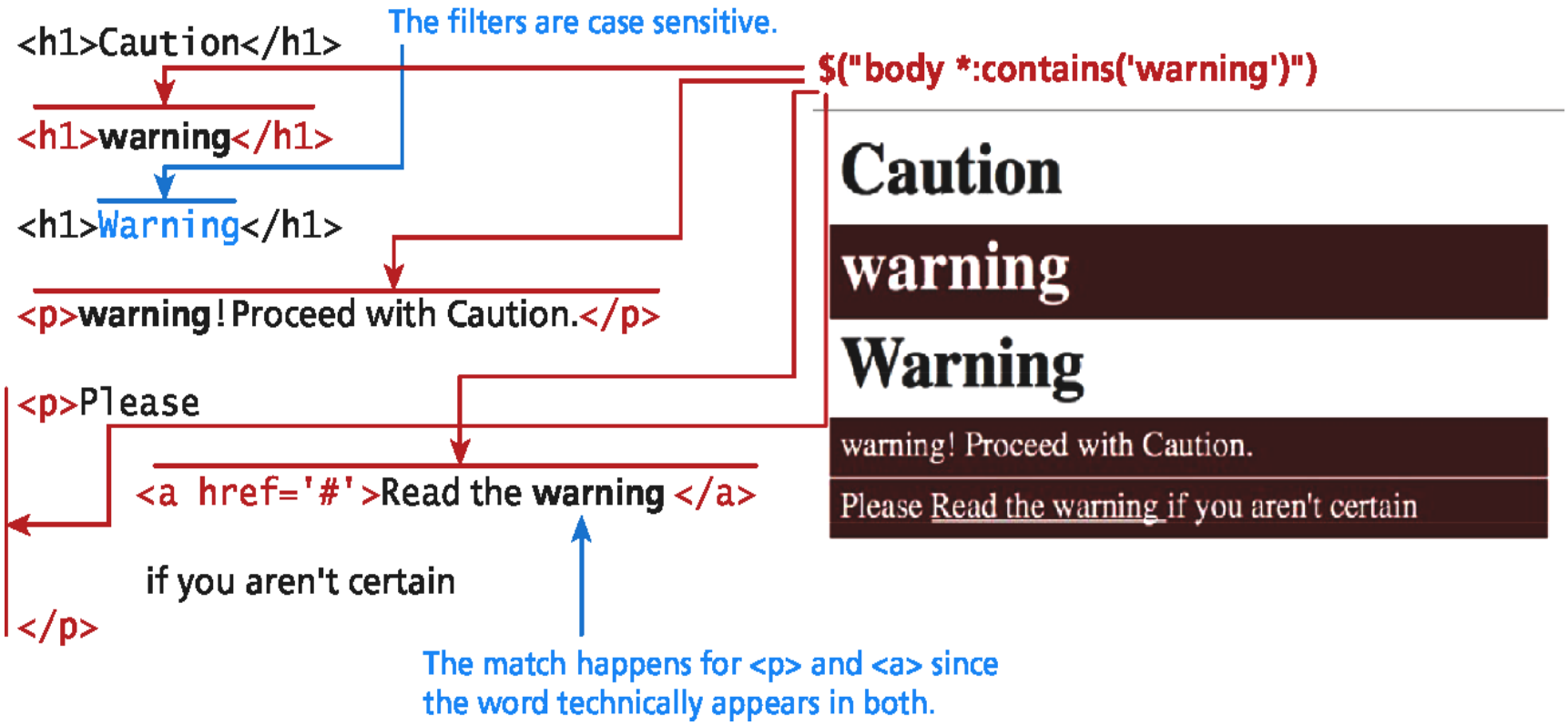
Select elements that have:

- a particular child using `:has()`
- have no children using `:empty()`
- match a particular piece of text with `:contains()`

Content Filters

Above and Beyond CSS

$("body *:contains('warning')")



Form Selectors

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all buttons
$(:checkbox)	$('[type=checkbox]‘)	Selects all checkboxes
$(:checked)	No Equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No Equivalent	Selects form elements that are disabled.
$(:enabled)	No Equivalent	Opposite of :disabled
$(:file)	$('[type=file]')	Selects all elements of type file
$(:focus)	$(document.activeElement)	The element with focus
$(:image)	$('[type=image]')	Selects all elements of type image
$(:input)	No Equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields
$(:radio)	$('[type=radio]')	Selects all radio elements
$(:reset)	$('[type=reset]')	Selects all the reset buttons
$(:selected)	No Equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements
$(:text)	No Equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input> fields with no type specified.

jQuery Attributes

Back to HTML now.

In order to understand how to fully manipulate the elements you now have access to, one must understand an element's *attributes* and *properties*.

The core set of attributes related to DOM elements are the ones specified in the HTML tags.

- The *`href`* attribute of an `<a>` tag
- The *`src`* attribute of an `<img>`
- The *`class`* attribute of most elements

jQuery Attributes

And some examples

In jQuery we can both set and get an attribute value by using the **attr()** method on any element from a selector.

```
//  link is assigned the href attribute of the first <a> tag
var link = $("a").attr("href");

// change all links in the page to http://www.ucd.ie/
$("a").attr("href","http://www.ucd.ie/");

// change the class for all images on the page to fancy
$("img").attr("class","fancy");
```

jQuery Attributes

Not only can you access them, you can also remove them by using
`removeAttr(attributeName)`

```
$("a").removeAttr("href");
```

Will remove the "`href`" attribute from `<a>`

HTML Properties

Full circle

Many HTML tags include *properties* as well as attributes, the most common being the *checked* property of a radio button or checkbox.

The `prop()` method is the preferred way to retrieve and set the value of a property although, attr() may return some (less useful) values.

```
<input class ="definition" type="checkbox"
checked="checked">
```

Is accessed by jQuery as follows:

```
var theBox = $(".definition");
theBox.prop("checked"); // evaluates to TRUE
theBox.attr("checked"); // evaluates to "checked"
```

Changing CSS

With jQuery

jQuery provides the extremely intuitive **css()** methods.

To ***get*** a css value use the css() method with 1 parameter:

```
$color = $("#colourBox").css("background-color"); // get the color
```

To ***set*** a CSS variable use css() with two parameters: the first being the CSS attribute, and the second the value.

```
// set color to red
$("#colourBox").css("background-color", "#FF0000");
```

jQuery each()

```
<table>
    <tr>
        <td>Row 1 cell 1</td>
        <td>Row 1 cell 2</td>
    </tr>
    <tr>
        <td>Row 2 cell 1</td>
        <td>Row 2 cell 2>/td>
    </tr>
</table>
```

jQuery each()

```
("td").each (function (index){
    console.log(index + "-->" + $(this).text());
});
```

Handling Events

Webpages are all about interaction

- Moving mouse, clicking on elements, typing
- Page is loaded, video is paused, etc
- Browsers announce things by ‘firing events’
- Developers can ‘listen’ to these events and react appropriately
- To listen to an event, a developer must register an ‘event handler’

Listening to Events

```
<button onclick="alert('Hello')">Say hello</button>
```

- Simplest way. Also, the worst way
  - Mixing of code and HTML
  - Duplication required wherever you want functionality
  - A slightly better way:

```
<button id="helloBtn">Say hello</button>
```

```
// Event binding using addEventListener
var helloBtn = document.getElementById( "helloBtn" );
helloBtn.addEventListener( "click", function( event ) {
alert( "Hello." );
}, false );
```

Does this work?

- On some browsers
- If you're using older browsers, such as IE, the standard `addEventListener` does not work
- Using jQuery allows you to NOT worry about browsers and versions

```
// Event binding using a convenience method
$( "#helloBtn" ).click(function( event ) {
alert( "Hello." );
});
```

jQuery Listeners

Set up after page load

You need to wait for the document to be fully loaded before you can add listeners to it.

With jQuery we wait for the document by using the $(document).ready() event

```
$(document).ready(function() {
  $("#toggleDiv").hide();
  $("button").click(function() {
    $("#toggleDiv").show();
  });
});
```

Equivalent

```
jQuery(function($) {
  // Run when document is ready
  // $ (first argument) will be internal reference to jQuery
  // Never rely on $ being a reference to jQuery in the global namespace
});
```

jQuery Listeners

Listener Management

While pure JavaScript uses the `addEventListener`() method, jQuery has `on()` and `off`() methods as well as shortcut methods to attach events.

`https://learn.jquery.com/events/event-basics/`

```
$(document).ready(function(){
   $(":file").on("change",alertFileName); // add listener
});
// handler function using this
function alertFileName() {
   console.log("The file selected is: "+this.value);
}
```

on and off examples

```
//Adding a normal click handler
$(document).on("click",function(){
    console.log("Document Clicked 1")
});
//Adding another click handler
$(document).on("click",function(){
    console.log("Document Clicked 2")
});
//Removing all registered handlers.
$(document).off("click")
```

Other API methods for DOM

- empty()
- closest()
- parents()
- parent()
- children()
- next()
- prev()
- filter()
- find()

More reading

- jQuery has much, much more. The more you explore the more you learn
- Resources
  - https://www.tutorialrepublic.com/jquery-tutorial/
  - `https://learn.jquery.com`

That’s all, folks!

Questions?