Javascript

Down the Rabbit Hole

Today's Agenda

- Javascript for loops with our checkbox list
- Javascript arrays
- Some more tooling around with Javascript
- Javascript Objects
- Javascript Functions – the weird & wild
- The Big Three (window, DOM, data)

for loops!

- Let’s finish looping through our checkbox list
- Remember:

```
for (var i=0; i < something; i++) {
        // Loops!
}
```

Arrays

- an object type that indexes data via index numbers
- *very* similar to Python's lists
- you can store any other types of objects inside of an array
- you access elements using [] notation

```
var materials = [‘hay’, ‘wood’, ‘hamster’]
materials[0] // will access hay
materials.pop() // Removes hamster
materials.push(‘kittens’) // Makes the list [‘hay’, ‘wood’, ‘kittens’]
```

Arrays cont.

// Our list is currently [‘hay’, ‘wood’, ‘kittens’]
materials.length // returns the length of the array
materials.indexOf(‘wood’) // Returns 1
materials.reverse() // Makes list [‘kittens’, ‘wood’, ‘hay’]
materials.shift() // Removes ‘kittens’ from list and returns it

Find more cool list things on W3C’s site. http://www.w3schools.com/jsref/jsref_obj_array.asp

Queue vs Stack

materials = [‘hay’,’wood’,’kittens’]

- Stacks are LIFO (Last in First Out)
  - materials.push(‘puppies’)
    [‘hay’, ‘wood’,’kittens’,’puppies’]
  - materials.pop() // Returns ‘puppies’
- Queues are FIFO (First in First Out)
  - materials.push(‘puppies’)
    [‘hay’, ‘wood’,’kittens’,’puppies’]
  - materials.shift() // Returns ‘hay’

List of Materials

Let’s modify our pricing pages so that the list is generated by a list, and when we add items to our page, it adds them to our list.

Objects

Objects group together a set of variables and functions in create a model.
At the end of the day, everything's an object in Javascript.

- in Python, we defined the model of our objects using Classes
- in Javascript, we can just make Objects.

Making an Object

```
var product = {
    checked: false,
    name: 'Wood',
    price: 15,
    stock: 10,

    adjustStock: function(num) {
        return stock -= num;
    }
};
```

Function Expression

```
var adjustStock = function(num) {
    return stock -= num;
}
```

- functions placed where an expression would normally be are called function expressions
- generally, you do not include a name for the function, making it an **anonymous function**
- function expressions are not evaluated until called – meaning you cannot call it before the interpreter reads it

Immediately Invoked Function Expressions (IIFE)

```
inStock = (function() {
    return stock > 0;
}());
```

- “iffy”s are function that are not given a name, as they are called immediately upon interpretation
- the variable they’re assigned to will be set to whatever they return
- Super important! Note the ()’s inside the closing)! This is what makes it an “iffy”!

When to use what

- Iffy's and Anonymous functions are best for when you need to run code only under very specific circumstances, rather than repeatedly from other parts of the script
- An object's methods are generally defined as Anonymous functions, as you only call them from within the object
- both can help reduce scope overlap
- Iffy's are helpful for when you want to wrap code even more securely

this

- a keyword commonly used inside functions and objects, which always refers to one object
- depending on where it's called depends on its value – it always refers to the default object at your current scope you're at

- *Global function:* **this** refers to the window object
- *Method inside an Object*: **this** refers to the containing object
- *Function Expression as a method:* **this** refers to the default object of the scope which it's called

The Big Three Objects

There are 3 sets of built-in objects in Javascript.

- Browser Object Model
  - a model of the browser/window the web page is loaded in
- Document Object Model
  - the DOM is a model of the current web page; it is a child of the Browser Object Model, as well
- Global Javascript Objects
  - these are not a single model; there are the various object types we've already been using (String, Number, Boolean, Object,

DOM Queries

Finding elements in the DOM can be tricky. There are a variety of ways to get them, though.

- getElementById(‘id): returns a single element with the id passed
- querySelector(‘css selector): returns the first element found with the CSS query

More DOM Queries

- getElementsByClassName(‘class’): returns an array of elements that have the passed class name
- getElementsByTagName(‘tagName’): returns an array of elements with the passed tagName
- querySelectorAll(‘css selector’): returns an array of elements that match the css query selector passed

Traversing the DOM

This is getting elements based on their relationship with other elements in DOM. The relations you can access by are:

- parentNode
- previousSibling
- nextSibling
- firstChild
- lastChild

Beware the Whitespace

Some browsers will make empty nodes between elements, especially when you add things via innerHTML. Beware of accidentally getting these empty elements and attempting to manipulate them.

innerHTML

- can be used on any element node of the DOM to both retrieve and replace content
- new content is provided as a string, which the browser will process into the DOM
- this reprocessing of the DOM can mess up event handlers and properties previously set
- innerHTML is also not as secure (more on that in a minute)

DOM Manipulation

- uses more code than innerHTML to accomplish the same things, and is slower for the browser to process
- more easily targets the individual nodes you want to affect
- allows direct manipulation of the elements you are creating, and their properties, before they're added to the DOM

innerHTML vs DOM Manipulation

+ can add a lot of new markup using less code
+ can be faster when adding a lot of new elements
+ it's a simple way to completely replace the content of one element
– should not be used to add content that has come from a user
– can be difficult to isolate single elements
– may break your event handlers

+ suited to changing one element from a DOM fragment with many siblings
+ does not affect event handlers
+ easily allows a script to add elements incrementally
– if you need to make a lot of changes, it's slower
– you need to write more code to achieve the same thing.

innerHTML's Security Issues

- innerHTML is more susceptible to Cross-Site Scripting (or XSS)
- XSS is when an attack places malicious code into your site
- This is a potential risk whenever you have data entering your site through
  - user's adding content through forms
  - multiple authors contributing content
  - third-party sites
  - files that are embeded or imported

innerHTML's Security issues

- XSS can let an attacker access info in your:
  - DOM
  - cookies
  - Session tokens (which could let them log into your site as your users)
- This things could allow the attacker to:
  - make purchases with the account
  - post defamatory content
  - spread their malicious code

innerHTML's Security issues

- Usually, the most dangerous bits are unvalidated and unfiltered code
- There are specific characters that when uncaught, can cause a lot of chaos
- You can use Javascript validation to ensure that only characters needed for your forms are submitted to your site
  - You'll also want to do server-side verification to make sure you filter out anything that may have gotten through, before you add it to your database

Page 228 - 231 in the text covers this all

Events!

- events are ‘fired’ or ‘raised’ when the event has *occurred*
- events ‘trigger’ a function or script when that function or script *reacts* to when that event is fired.
- There are several types of events:
  - UI, keyboard, mouse, focus, form, and mutation

UI Events

Occur when a user interacts with the browser's UI rather than the web page.

- load: web page has finished loading
- unload: web page is unloading (usually cause a new page is loading)
- error: browser encounters a JS error or asset doesn't exist
- resize: window has been resized
- scroll: window has been scrolled

Keyboard Events

Occur when a user interacts with the keyboard (what a surprise!)

- keydown: user first pressed a key (this repeats while the key is still depressed)
- keyup: user released a key
- keypress: character is being inserted (repeats while the key is depressed)

Mouse Events

- click: pressed and releases a button over the same element
- dblclick: double click on the same element
- mousedown: user pressed a mouse button over an element
- mouseup: user releases a mouse button over an element
- mousemove: user moved the mouse
- mouseover: user moved the mouse over an element
- mouseout: user moved the mouse off of the element

Focus Events

focus/focusin : element gains focus
blur/focusout : element loses focus

Form Events

Occurs when a user interacts with a form element.

- input
- change
- submit
- reset
- cut
- copy
- paste
- select

Mutation Observers

Occurs when the DOM is changed by a script. This could happen when something is added or removed from the DOM. These replaced Mutation Events, as instead of releasing a bunch of events for a change, they fire once, after everything has finished being manipulated.

Event Handling

1. Select the element (or the window) that you want to script to respond to
2. Declare which event on the selected node(s) will trigger a response
   a. this is called **binding** in the programming world
   b. keep in mind that some nodes will not trigger some events
3. State the code you want to run when the event occurs, usually by declaring which function (anonymous or named) should be called

How to Bind an Event to an Element

HTML Event Handlers!

- These are a **bad idea** to use, as they tightly couple the HTML to the Javascript
- You might still run into them (slash use them to learn JS ;D)

```
<button onclick=”doAThing()”>Button!</button>
```

How to Bind an Event to an Element

DOM Event Handlers!

- These are a good replacement to HTML Event Handlers, and are even named similarly!
- Allow you to decouple the HTML from the JS
- Can only call one function (sadface)

```
button.onclick = doAThing();
```

How to Bind an Event to an Element

DOM Level 2 Event Listeners

- released in 2000, these are the favored way to handle events
- allow for multiple functions to be triggered with one event
- this reduces the number of conflicts that might occur in the same script
- sadly, this does not work in IE8 or earlier

Event Listeners

element.addEventListener(‘event’, functionName [, flowBool]);

button.addEventListener(‘click’, doAThing, false);

To have multiple things trigger off an event, similar add another listener that calls another function.

Removing an Event Listener is very similar to adding it, but you use removeEventListener(), which has the same parameters.

Event Listeners

Important! You cannot add the () after the function name in the addEventListener parameters, because that would make the browser run the function when it interprets it (oops!).

To pass parameters to a function triggered via an event listener, use Anonymous Functions:

```
button.addEventListener('click',function() {doAThing(5);}, false);
```

Event Flow

By default, an event trigger's outward, and *bubbles* up. This is flow = false.

Setting an event's flow to true will mean it will go from least specific to most, and be *captured*. Capturing is not supported on IE8 or earlier.

Event Object

- The Event object tells you information about the event, and the element the event happened on.
- it is passed to any function that is the event handler or listener
- it is passed automagically, so your functions which are triggered by an event need to have a parameter for the event object to be passed into

```
function doAThing(e) {
    var target = target;
}
button.addEventListener('click', doAThing, false);
```

Event Objects with Anon

```
function doAThing(e, 5) {
   var target = e.target;
}

button.addEventListener('click', function() {
     doAThing(e, 5);
}, false);
```

Event Delegation

- Creating event listeners for a lot of elements can slow down a page, but event flow (bubble and capture) allows you to listen for an event on a parent
- Being more general with your Event Listeners is better!
  - write less code
  - get better performance
  - automagically works with new elements
- solves some limitations with using this as a target
- simplifies your code (zomg, so much)

Modifying Event Objects

preventDefault() – prevents an event's default behavior from occurring, such as a link event that would take a user to a new page

stopPropagation() – stops the event from bubbling up, so it won't trigger other event handlers

More Events?

Chapter 6 in Javascript & JQuery is all about events. These slides cover up to page 267. If you want to see more examples, or learn more about specific event types, or using this instead of the event object, read pages 268 –292

javapic

Congrats! You've been hired to work at javapic, the latest and greatest startup in Portland! We're a small company that focuses on developing and enriching cloud-based communities by empowering visual storytelling.

Our site is pretty gorgeous right now, but we're missing some pizazz. That's where you come in! You've been hired to add Javascript to our pages.

We want a Jumbotron that automatically loops through some of our images every 20 seconds, a login form that won't let users continue until their inputs are correct, and a gallery that is auto-populated with all the images in the folder. When an image is clicked, it should show up larger, and clicking anywhere on the page should remove the larger preview.