Introduction to Computer Science I
COMP 2406A – Winter 2020

Javascript and the Document Object Model

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Understand the document object model

Use Javascript to read information from a webpage

Understand the concept of event handlers

Create reactive web pages using Javascript

Before We Start

An important note before we start – web development involves both a client AND a server

Keep this in mind throughout the course. It may be unclear at first, but will become second nature with practice

Before We Start

The examples we work on today will only involve client-side programming

We will be manipulating the HTML that the client would generally receive from some server

Since we don't have a server currently, we will manipulate HTML files stored locally

Before We Start

Another important note – we will be doing a lot of event-based programming in the course

This is a different way of thinking than you may be used to.

We define the events that may occur and the code that should be used to handle those events (e.g., user clicks, key presses, etc.)

Before We Start

Most of us (if not all of us) use the web quite a lot

Throughout the course, take some time to think about what is happening while you browse

Relate the concepts in the course to what happens on a web page. Try to understand what approaches may be involved in the implementation.

Before We Start

There are MANY different HTML components, CSS styles, Javascript events, etc.

Don't try to memorize them all – it isn't that important.

Instead, focus on the concepts – if you can browse the documentation and find a solution to your problem, you'll be fine

Before We Start

There are also MANY different ways to solve the problems we will discuss

Think before you program, compare/contrast possible solutions

Some ways are easier/better than others and you can save yourself a lot of time

Intro to Javascript

Last time, we saw how to use Javascript to do some basic programming

Now, we will use Javascript for what it was originally designed for: dynamic web pages

Intro to Javascript

Javascript was originally intended to be used for both client- and server-side development (and it is now, as we will see soon)

To start, we will focus on strictly client-side programming

We will make web pages react and change based on user interaction

Event-Based Programming

We will be doing a lot of event-based programming in this course

A user makes a request, we process it and respond

Anybody who has done Java GUI programming?

A Basic Web Page

Consider the basic web page 02-basic.html

When we load the page in a browser, it is parsed and displayed on the screen

The browser uses an underlying model to represent the structure, called the DOM

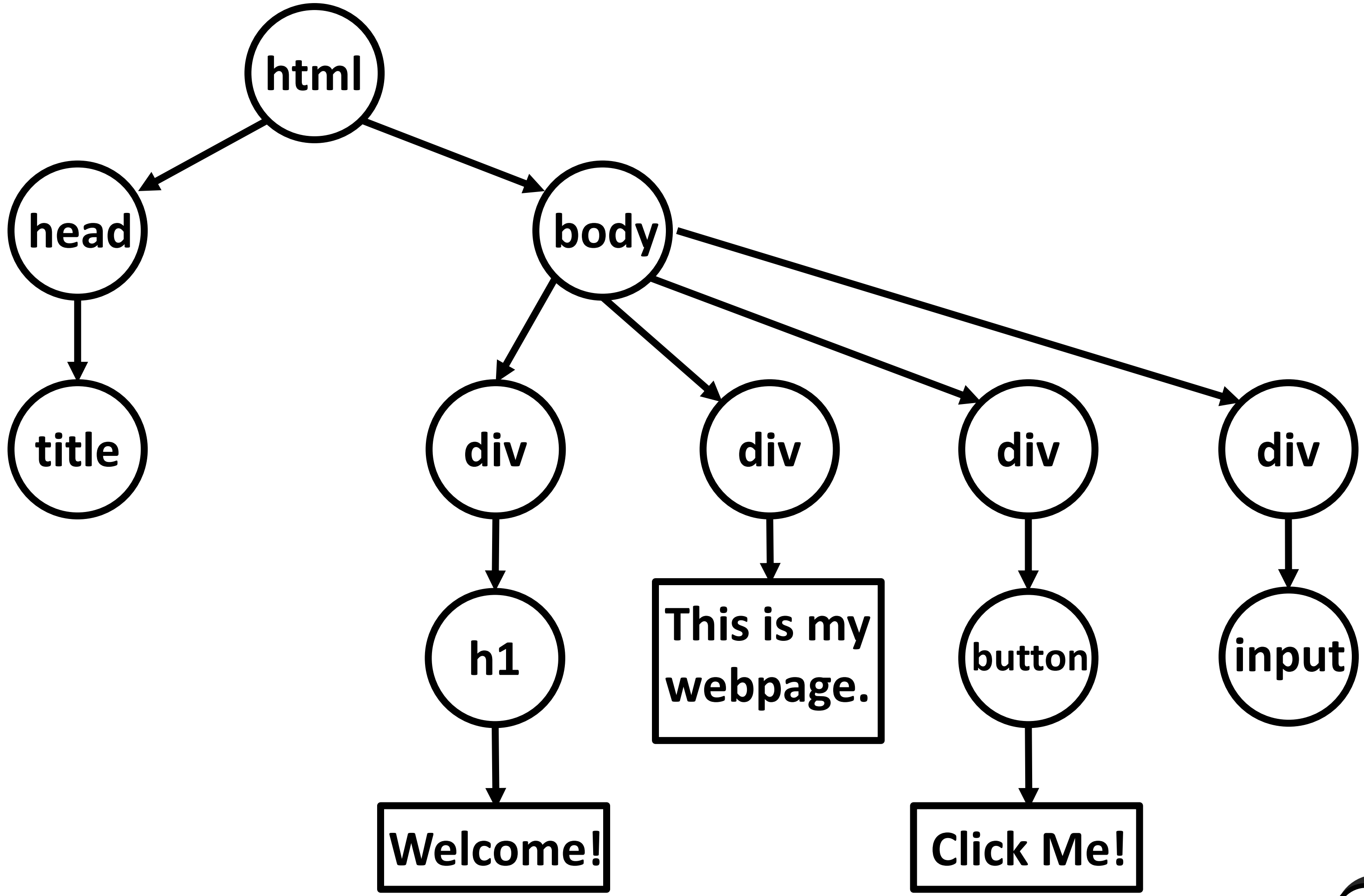
Document Object Model

DOM: Document Object Model

A W3C standard that provides an interface to dynamically modify a page displayed in a browser

The DOM uses a tree structure

DOM Tree of 02-basic.html



Javascript and the DOM

We discussed running Javascript in the browser

Using Javascript, we can:

Change HTML elements, attributes, and CSS styles

Add/remove HTML elements

Respond to actions on HTML elements (clicks, etc.)

Create new events to respond to

The ‘document’ Object in Javascript

Javascript code in the browser has access to a ‘document’ object

This object represents the root node of the DOM tree

Finding an Element

In order to add/remove/modify anything, we need to first get an element from the page

Everything on a web page is stored in some element

By using the ‘id’ HTML attribute within elements, we give them a unique name that we can use to refer to that particular element

Finding an Element

The document object allows us to call:

document. getElementById(string)

Returns the element on the page with the given ID, or null if no matches are found

Once we have a variable referencing an element, we can do many things

A First Modification

One thing we can do is set the inner HTML content of an element

```
let someEle = document.getElementById(“main”);
someEle.innerHTML = “Any HTML you want”;
```

Add some script to the 02-basic.html page to modify one of the elements inner HTML

Responding to Events

This example isn’t all that impressive – we could have just set that HTML originally

In general, we will respond to events and handle those events by making changes

There are various event types…

Browser Event Types

You can handle events such as:

Clicks

Loading/unloading the page

Loading an image

Mouse movement

Input fields change, keyboard buttons

Form is submitted

(as usual, w3schools.com has an in-depth list)

The ‘onclick’ Event

One of the most common events is the ‘onclick’

This is triggered when one of the HTML elements is clicked on by the user (e.g., a button)

You can add a Javascript function to handle this event within the HTML specifying the element

Add an onclick handler for the button in 02-basic.html

The ‘onclick’ Event

Since onclick is an attribute of the element, you can also add a handler through Javascript

If someElement is a variable representing an HTML element, then:

```
someElement.onclick = someFunction
```

(note: no parenthesis in this case)

The ‘onload’ Event

The ‘onload’ event is triggered when an element on the page is loaded by the browser

Typically used on the <body> tag for initialization

There is also an ‘onunload’ event, when the user leaves the page

The ‘onchange’ Event

The ‘onchange’ event is triggered when:

A radio button is checked/unchecked

A checkbox is checked/unchecked

Text and other components lose focus (after being modified)

Typical use: validating inputs or selections

Mouse Events

There are several mouse-based events:

onmouseover – mouse enters space of element

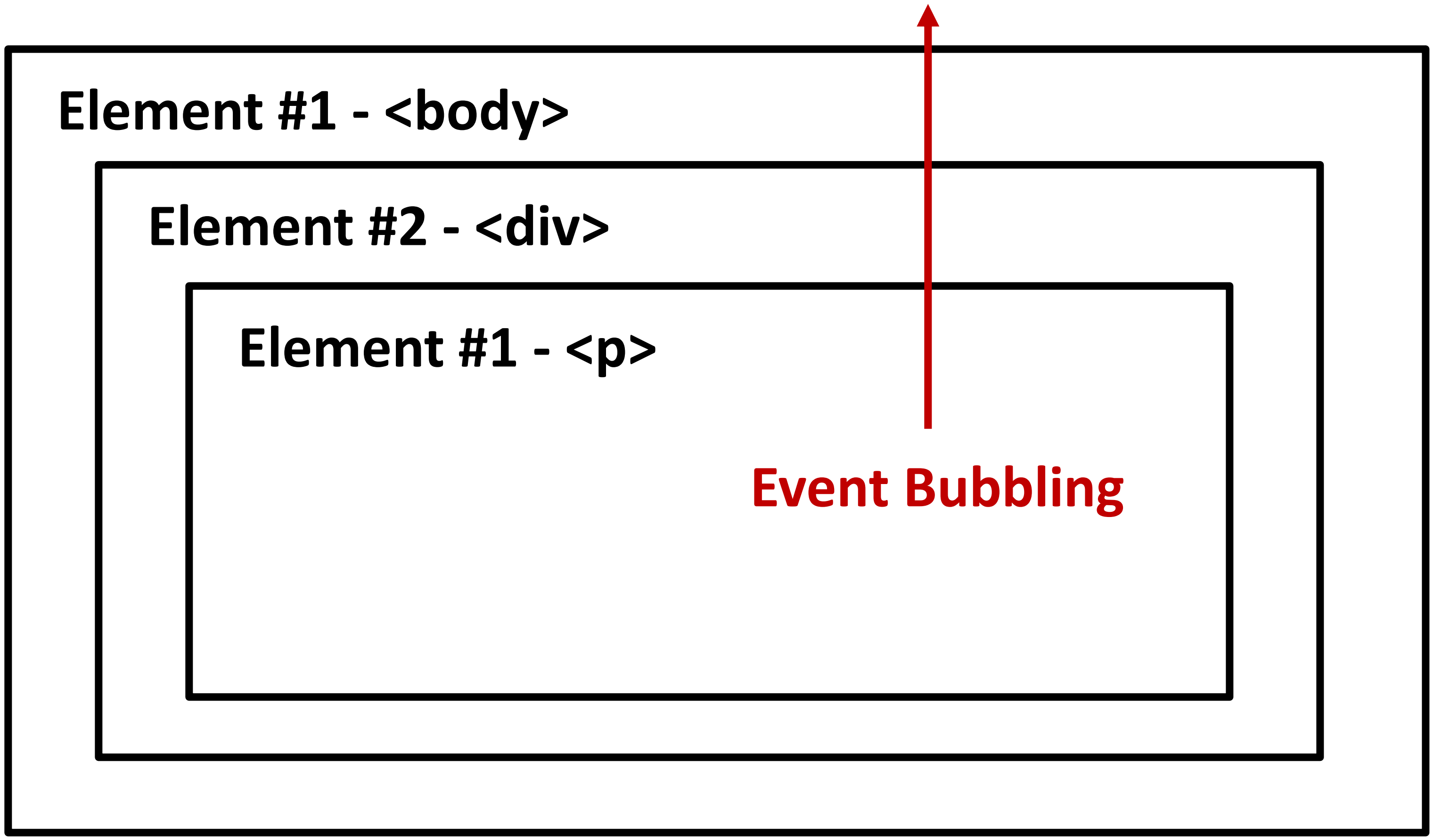
onmouseout – mouse leaves space of element

onmousedown – mouse button is pushed down

onmouseup – mouse button is lifted

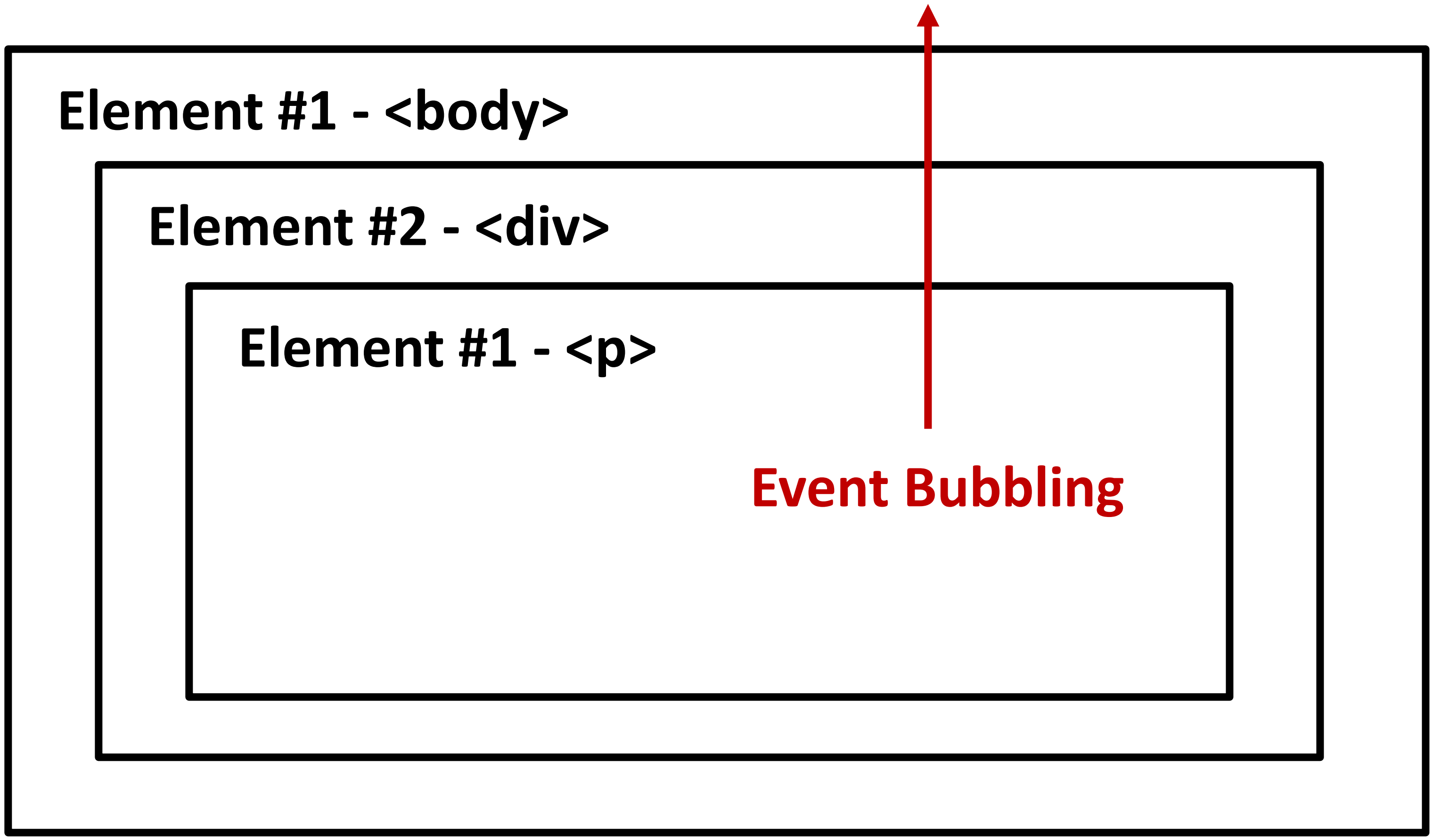
onclick – when element is clicked

Event Propagation



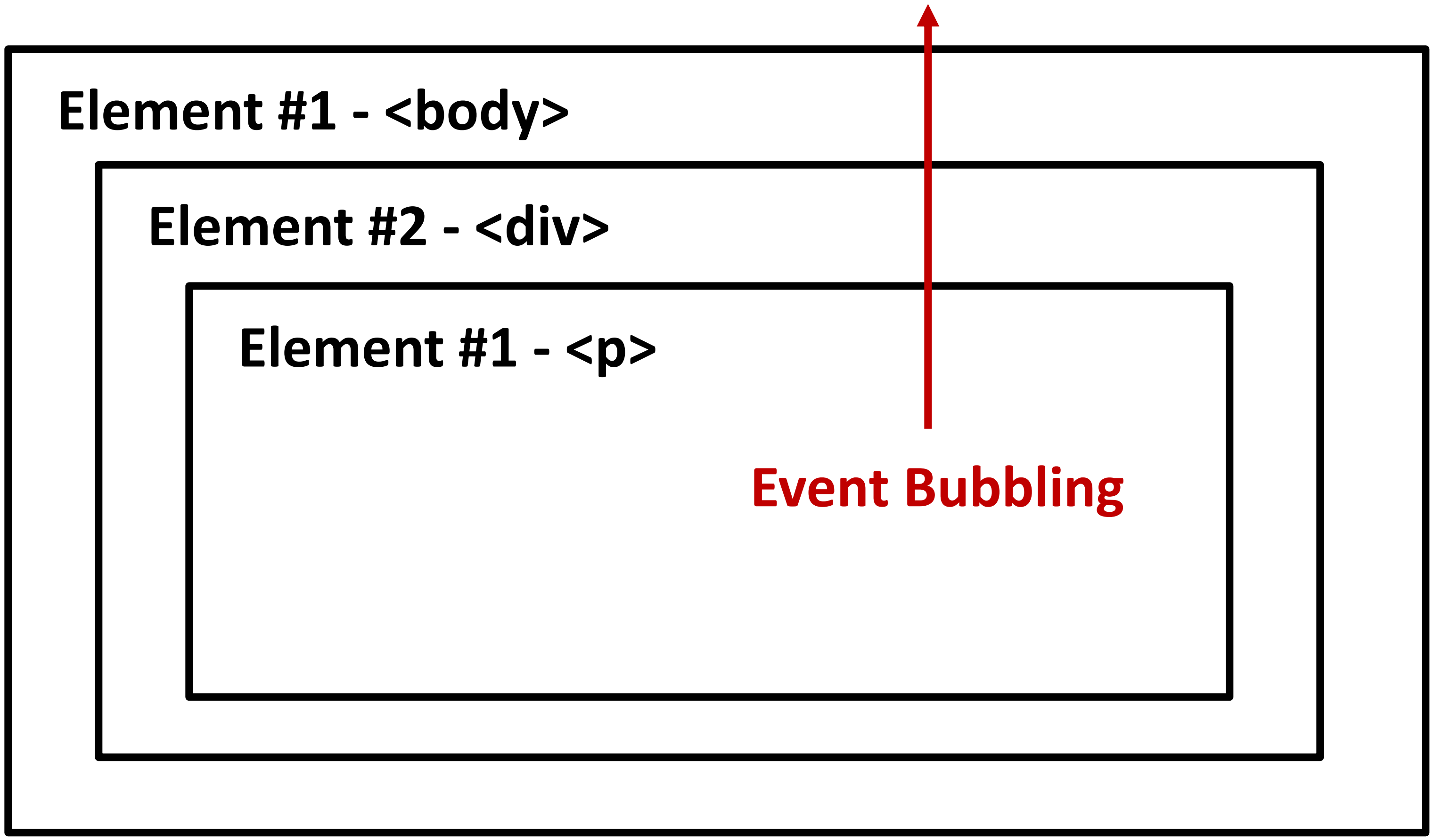
Events ‘bubble’ up the DOM tree to outer elements

Event Propagation



You can stop the bubbling with event.stopPropagation()

Event Propagation



Note: if a handler is not registered, nothing happens

Another Way to Add Handlers

There is an additional way to add event handlers:

someElement.addEventListener(String, Function);

String – name of the event (click, load, no ‘on’)

Function – the function to call for that event

This allows us some flexibility…

Another Way to Add Handlers

We can call addEventListener more than once for the same element – giving us multiple handlers

We can add handlers to a page, even if we didn't write the HTML (e.g., we loaded it from elsewhere)

We also separate the display content (HTML) from the behaviour (Javascript handlers)

Another Way to Add Handlers

You can also remove event handlers:

someElem.removeEventListener(String, Function)

String – type of event (click, load)

Function – the handler function to remove

This gives us even more flexibility – we can change how things are handled dynamically as we need to

A Note on Events

Not all events are supported by all browsers

Each has its own Javascript engine

Older browsers may not support all functionality we talk about (here, we assume recent browser is used)

Another Important Note

Remember Javascript is single-threaded!

Your handler functions should run quickly or make use of asynchronous calls

If not, your page may seem broken

Modifying CSS Style

By retrieving an HTML element from the page, we gain access to that elements attributes

This includes its ‘style’ attribute object, which determines the look of the element

In general: someElem.style.someProp = “something”

Modifying CSS Style

Example – changing the color of an element to red:

someElement.style.color = “red”;

Example – hiding/showing an element:
someElement.style.visibility = “hidden”
(there is also a ‘display’ property)

In general, any CSS property can be modified

Navigating the DOM

The DOM is a tree model

Each node has a single parent (except root)

Nodes can have 0+ children

Types of Nodes

There are different types of nodes in the DOM:

Element nodes – for HTML elements

Text nodes – the text of the ‘inner HTML’

Navigating the DOM

From any one element, you can access properties for:

parentNode

childNodes (array)

firstChild

lastChild

nextSibling

previousSibling

Allows us to move through the DOM systematically

Navigating the DOM Example

Consider the 02-checkboxes.html page

Add a handler that counts how many of the checkboxes within the “boxes” div are selected

Note: this is done in a general sense, so if we add/remove checkboxes, it will still work (important with next slides)

Adding New Elements

We can use the document object to create new elements

```
let para = document.createElement(“p”);
let text = document.createTextNode(“words!”);
para.appendChild(text);
let elem = document.getElementByID(“someDiv”);
elem.appendChild(para);
```

What is this block of code doing?

Adding New Elements

elem.appendChild(node) adds to the END of children

You can also add at a specific location using:

elem.insertBefore(newNode, childNode)

This inserts newNode just before childNode in elem’s children (if newNode is already a child, it is just moved to before childNode)

Removing Children

You can also remove a child element:

```
parentElem.removeChild(childElem);
```

What if we don't know the parent node?

Removing Children

If we don’t know the parent node, we can find the child we want to remove and use its ‘parentNode’ property

Replacing Children

Finally, you can replace a child:

```
parentElem.replaceChild(newChild, oldChild);
```

All of these methods combine to allow us to manipulate the HTML in any way we want

Additional Selectors

In addition to getting an element by ID, you can get a collection of elements

document.getElementsByTagName(string)

This finds all elements of type *string* in the document (e.g., “p”, “div”, etc.)

Returns an HTMLCollection – like an array (.length property), but NOT an array (no push/pop/etc.)

Additional Selectors

You can find elements by CSS class name:

someElem.getElementsByClassName('class1 class2');

Starts search with someElem as root

Can include multiple classes – treated as an AND operation

Additional Selectors

You can find elements by HTML ‘name’ attribute:

```
document.getElementsByName(string);
```

Gets all elements with the given name attribute

Summary

So we can decide what events to respond to and how

We can read information from the webpage

We can modify the HTML in any way we want

This allows us to make dynamic client-side applications

Summary

An important requirement for all of this to work is structured data

We need to have a naming scheme so we can retrieve specific elements

This is why thinking/designing before programming is important

Summary

Working with data in a structured manner (naming schemes and protocols) will be a fundamental idea throughout the course

Again, this is extremely important within the context of web development

Summary

We are currently limited to using information stored in the page or our script

Soon we will discuss development on the server-side

This will give us more power, as all the data necessary does not need to be sent/stored on the client

Questions?

Questions?