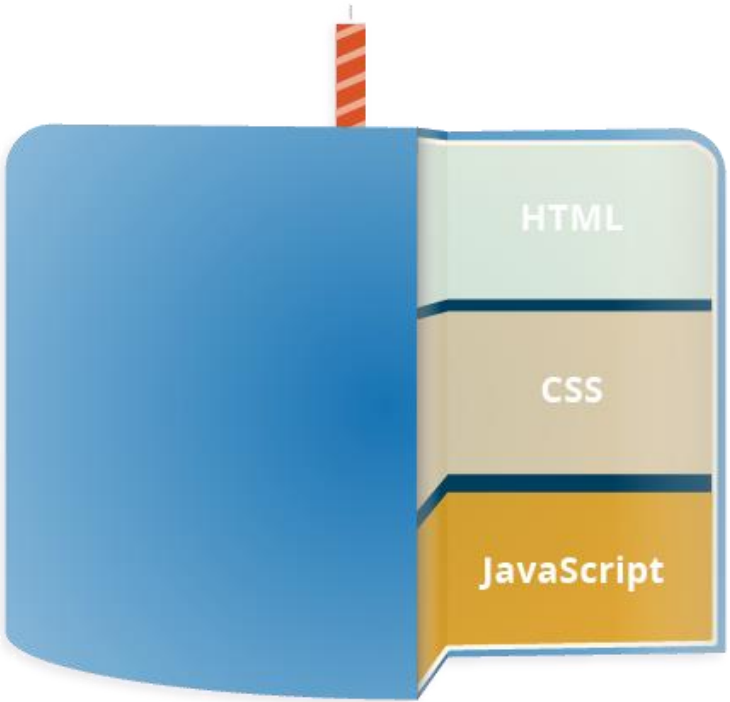
Week 3: Javascript & the DOM

INFSCI 2560
Web Technologies & Standards

Today's Agenda

- Introduction to JavaScript
- Break
- Introduction to the DOM
- Assignment 1
- Activity 3

How HTML, CSS, & JS fit together



Like a Cake!

- HTML is the *content layer*
  - Structure and Semantics
- CSS is the *presentation layer*
  - Visual Display in context
- JavaScript is the *Behavior layer*
  - Transformation and Interactivity

This is called the *separation of concerns*

Image from [MDN - What is JavaScript]

JavaScript

```
//access content
document.getElementsByTagName(“h1”);

//modify content
document.write(“Hello world”);

//program behavior
myElement.style.color = “blue”

//react to events
document.addEventListener(“click”,
function(event){
    Alert(“You clicked!”)
});
```

- Access the content
  - You can select Elements and their content and store pointers or references to them in JavaScript variable.
- Modify the content
  - All HTML Elements are JavaScript objects with properties and methods for changing them
- Program behavior
  - You can use JavaScript to programmatically change the content and structure of the web page
- React to events
  - You can write JavaScript functions that will be triggered by specific browser events

JavaScript runs where it is located

```
<!DOCTYPE html>
<html>
<head>
<script src=”MyScript.js”></script>
<script>
   function myFunction() {
     Console.log("Paragraph changed.");
   }
</script>

</head>
<body>
  <h1>A Web Page</h1>
  <p id="demo">A Paragraph</p>
  <button type="button"
     onclick="myFunction()">Try it</button>
  <script>var x = “hello world”</script>
</body>
</html>
```

- You can put script tags in many places In HTML documents
  - Inside a <script> element or in an external file
  - In the <head>
  - In the <body>
- It runs when loaded, from top to bottom
  - Will halt the page loading
- Best practice is loading an external file
  - Separation of concerns
  - Easier to read/maintain
  - Can be cached for performance

Brief introduction to JS as a programming language

About JavaScript

- Just-in-time compilation (JIT)
- Most popular scripting language for the web
  - And now on the server
- A language developed at Netscape (anyone remember that?)

We aimed to provide a "glue language" for the Web designers and part time programmers who were building Web content from components such as images, plugins, and Java applets. We saw Java as the "component language" used by higher-priced programmers, where the glue programmers—the Web page designers—would assemble components and automate their interactions using [a scripting language].

- Standardized as ECMAScript in 1996.
  - Latest stable release is ECMAScript 2021. ECMAScript 2022 is coming.
- *JavaScript* is to *Java* as a *Hamster* is to *Ham*

Statements

```
// Four Statements
var x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4

// Single line statement
x = 5; y = 6; z = x + y;
```

- The basic blocks of Javascript code
- Composed of
  - Values
  - Operators
  - Expressions
  - Keywords
  - Comments
- Keywords - the reserved words of the JavaScript language
- Statements are separated by a semicolon
  - Although you can forget it and the code will still work

Javascript Keywords

break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements

Comments

```
// Function: creates a new paragraph and
// append it to the bottom of the HTML body.
function createParagraph() {
  let para = document.createElement('p');
  para.textContent = 'You clicked the button!';
  document.body.appendChild(para);
}

const buttons =
document.querySelectorAll('button');

for (let i = 0; i < buttons.length ; i++) {
  buttons[i].addEventListener('click',
createParagraph);
}
```

- Use // for one line comments
- Use /* comment */ for multi-line comments

Code from [MDN - What is JavaScript #Comments]

Variables

```
// JS is a dynamic language
// no type declarations
var x = "string" // no type declaration
var x = 5.6 // variables can be
// optional initialization
var x; // == undefined
x = 23 // initialize later
// Scope
if (true) {
  var x = 5;
}
console.log(x);  // x is 5
// let behaves differently
if (true) {
  let y = 5;
}
console.log(y);  // ReferenceError: y is not
defined
```

Rules for variable name

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

Three ways to declare

- **var** - old way, global scoping
- **let** - declares within a specific scope (in a function, object, or global), more strict
- **const** - for creating immutable variables

For more: MDN - Javascript Guide: Grammar and Types#Declarations

Data types

```
var a = true; // boolean true
var b = null; // null
var c; // undefined
var d = 3; // integer number
var e = 3.4; // floating point
var f = "hello world"; // string
var g = Symbol(42) // Symbol type
var h = Object() // generic object
```

Six primitive data types

- Boolean - “true” and “false”
- Null - nothingness
- Undefined - declared with no value
- Number - integers or floating points
- String - sets of characters
- Symbol - a newly added type for creating unique values. Often for object properties

And the Object data type (will discuss later)

Collections

```
var arr = Array(1,2,3) // array
constructor
var arr = [] // empty array
var coffees = ['French', 'Bad','Diner'];
// array literal
// multiple types and empty
var things = ['bad', 3, Object(), ,
[1,2,3], `hello`];
//indexing
thing[0]; //'bad'
thing[things.length - 1] // hello
```

- Two main data structures for containing data
  - Array - Square brackets
  - Object - Curly brackets (looks like JSON)
- Arrays are just objects, with a special syntax for literal declaration
- Arrays are an *ordered* set of values
  - Numeric index starting at zero
- Access members via numeric index
- Objects are pairs of names and associated values
- Accessed via name via bracket or dot notation
- Both can contain arrays or objects as values to create nested, complex structures

For more: MDN - Javascript Guide: Indexed Collections

Collections

```
things[0] = 'good'; // change ‘bad’ to ‘good’

//Object literals
var car = {make:"honda", model:"Fit", year:2012}

//accessing
car.make;
car['model']; car.model = 'RAV4'
car
```

For more: MDN - Javascript Guide: Indexed Collections

Functions

```
// function declaration
function doubler(x) {
  return x * 2;
}
doubler(2); // 4

// function expression
var doubler = function(x) {
  return x * 2;
}
doubler(2); // 4
```

- Fundamental building blocks of JavaScript
- Declaring functions
  - Name of function
  - Parameters in parentheses separated by commas
  - JavaScript statements in curly brackets { }
  - Use “return”
- Call functions with name(argument)
  - Parameters when declaration
  - arguments when calling

For more: MDN - JavaScript Guide: Functions

Functions

```
//return statement
function boo(x){x * x}; boo(3) //
undefined

function boo(x){return x * x};
boo(3); // 9

//anonymous functions - Callback
document.addEventListener("click",
function(e) {
  console.log("You clicked" + e)
});
```

- Function Expression
  - Giving names to call within the function
  - Anonymous functions
  - For passing functions as arguments

Hoisting

```
// variable hoisting with let and var
console.log(y); // undefined
var y = 4;
console.log(x); // ReferenceError
let x = 3;

foo(); // "bar"
// function declaration
function foo() {
  console.log('bar');
}
// Function expressions must happen in order
baz(); // TypeError: baz is not a function
// function expression
var baz = function() {
  console.log('bar2');
};
```

- Variables and Functions can be *hoisted*, that is, referred before they are declared
- Variables will be “undefined” until they are initialized
  - If you use “var”, “let” behaves differently
- If you *declare* a function,when you start with “function…”, you can call the function before the declaration
  - The function is *hoisted* to the top
- Function expressions, when you start with “var f = function…”, cannot be called
  - The function is not hoisted to the top

For more: MDN - Javascript Guide: Grammar and Types#Variable Hoisting

Operators

```
// assignment
var x = 5;
x += 2; x -= 1; // x is 6. increment operator
// comparison
x == 6 // true
x != 6 // false
x === 6 // strict equality, use this!
x > 10; x < 10; // false. true. comparison
x + 1; x - 10; x / x; x * x; // standard math
operators
1 / 2 == 1.0 / 2.0; // true
// logical
true && true; true && false; false || true; //
true. false. true.
// string
"Hello" + " " + "World!"; // "Hello World!"
```

- Assignment (=)
  - Assigning variables
- Comparison
  - Checking similarity
  - Use ===!
- Arithmetic (+, -, *, /, %, ++, --, **)
  - Doing maths
- Logical (&&, ||, !)
  - Checking truth values
- String (+)
  - For concatenating strings

For more: MDN - Javascript Guide: Operators

Conditional statements

```
if (x < 10) { // is x less than 10?
  x++; // increment
} else {
  console.log("Hit 10")
}
// Multiple conditions
if (x === "cat") {
  console.log("meow" + "I like "+ x + "s");
} else if (x === "dog") {
  console.log("Woof" + "I like "+ x + "s");;
} else if (x === "hamster") {
  console.log("squeak" + "I like "+ x + "s");;
} else {
  console.log("I like "+ x + "s");;
}
```

If statements evaluate the truthyness or falsyness of a statement.

The following values evaluate to false (also known as Falsy values):

- false
- undefined
- null
- 0
- NaN
- the empty string ("")

All other values evaluate to true when tested in a conditional statement

For more: MDN - Javascript Guide: Control Flow and Error handling

Loops

- Many different ways to loop in Javascript
  - For
  - Do...while
  - While
  - For...in
  - For...of
- Use the “break” statement to terminate the loop completely
- Use the “continue” statement to terminate current iteration and go to the next iteration

Note: Loops will halt browser execution until complete. Javascript is not multi-threaded!

For more: MDN - Javascript Guide: Loops and iteration

```
// generic for loop
var step;
for (step = 0; step < 5; step++) {
  // Runs 5 times, with values of step 0
through 4.
  console.log('Walking east one step');
}
var n = 0;
var x = 0;
while (n < 3) { // while condition is
true
  n++;
  x += n;
}
```

Loops

```
// use for...of to iterate elements
var animals = [‘frog’, ‘dog’, ‘cat’, bird’];

for (var animal in animals) {
   console.log(animal); }

for (var i of arr) {
   console.log(i);
}
```

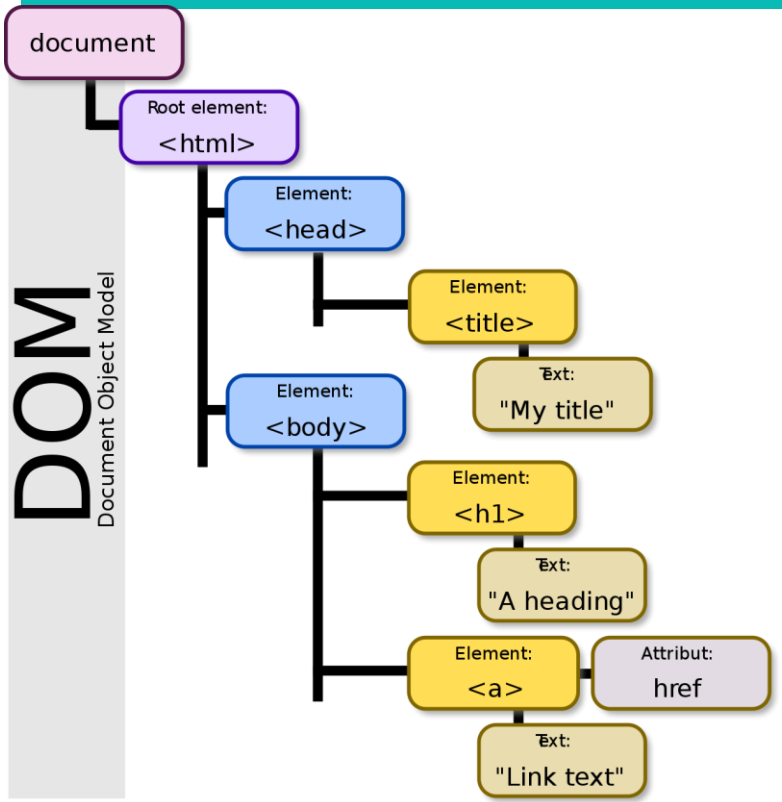
Built-in Objects

- Browser object model (BOM)
  - Browser specific object hierarchy for accessing information about the window (“window” object)
- Global Javascript objects
  - Strings, Numbers, Math, Date...And many more (See MDN Docs)
- Document object model (DOM)
  - Thats coming up next!

Break

The Document Object Model (DOM)

What is the DOM?

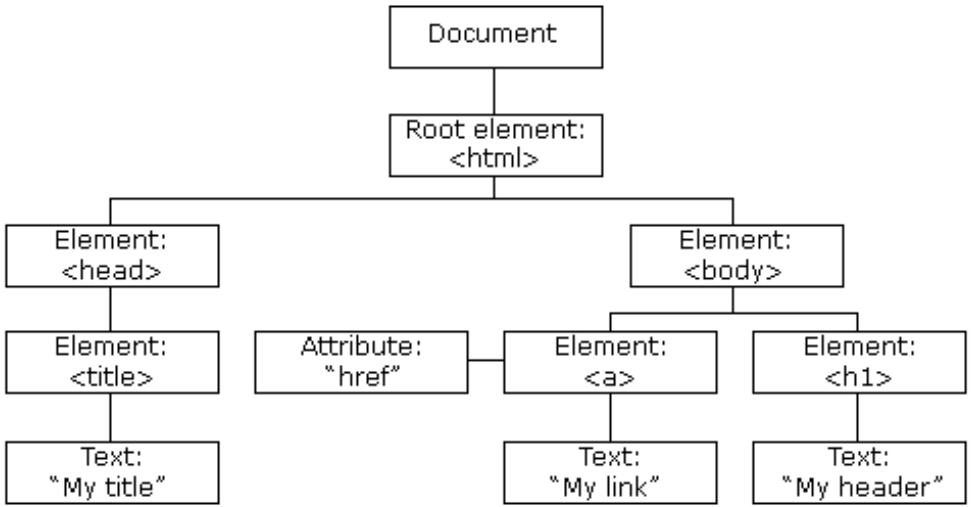


- A W3C Standard for accessing documents

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

- Not part of the HTML or JavaScript Standards
- Rules for how to *represent* HTML documents and
- An API for Javascript to access & change the contents of a web page

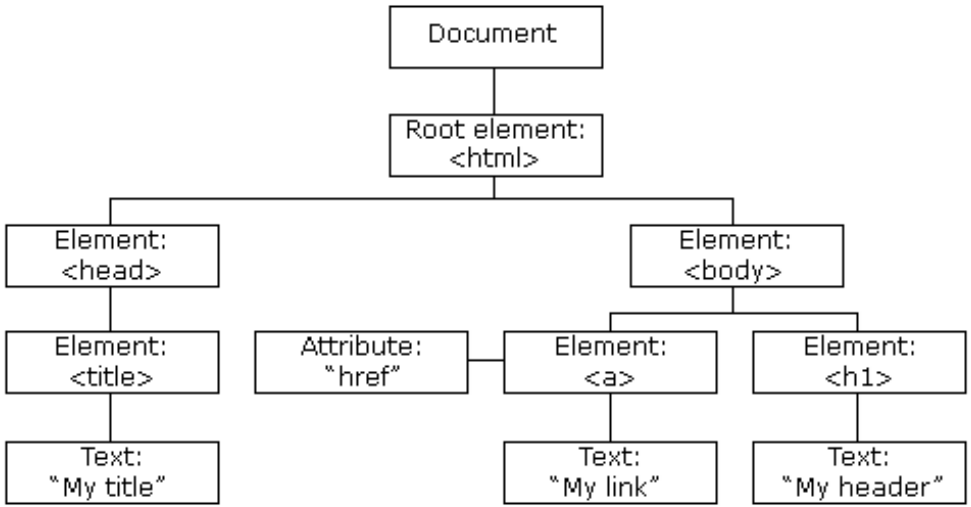
The DOM is a tree



- **Element node**: An element, as it exists in the DOM.
- **Root node**: The top node in the tree, which in the case of HTML is always the `HTML`node (other markup vocabularies like SVG and custom XML will have different root elements).
- **Child node**: A node *directly* inside another node. For example, `A` is a child of `BODY` in the above example.

Content from [W3C Schools - JavaScript HTML DOM]

The DOM is a tree



- **Descendant node**: A node *anywhere* inside another node.
- **Parent node**: A node which has another node inside it. For example, `BODY` is the parent node of `H1` in the above example.
- **Sibling nodes**: Nodes that sit on the same level in the DOM tree. For example, `A` and `H1` are siblings in the above example.
- **Text node**: A node containing a text string.

Content from [W3C Schools - JavaScript HTML DOM]

What a document looks like in HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Simple DOM example</title>
  </head>
  <body>
      <section>
        <img src="dinosaur.png" alt="A red Tyrannosaurus Rex: A two legged
dinosaur standing upright like a human, with small arms, and a large head
with lots of sharp teeth.">
        <p>Here we will add a link to the <a
href="https://www.mozilla.org/">Mozilla homepage</a></p>
      </section>
  </body>
</html>
```

Code from [MDN - Manipulating Documents#The document object model]

What the same document looks like in DOM

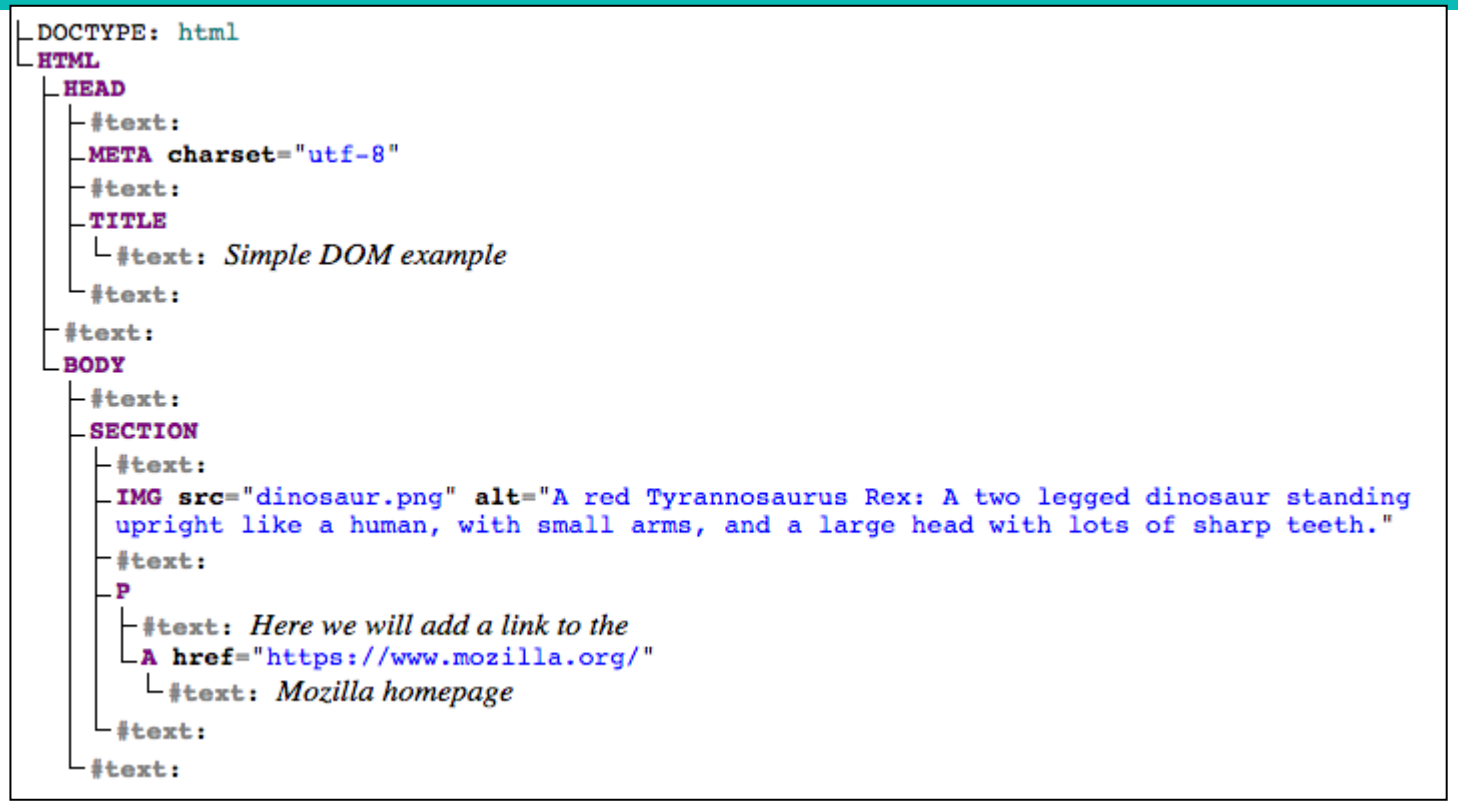


Image from [MDN - Manipulating Documents#The document object model]

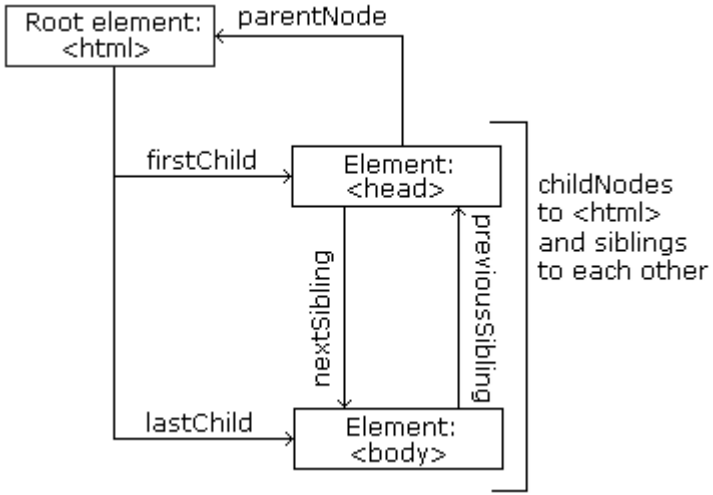
Accessing DOM Elements in JavaScript

```
// Grab a single element via id attribute
var myIdElement = document.getElementById('id')
// Grabs a single element via CSS query
// Select <p> whose parent is a div with class "bar"
var myElement = document.querySelector('p > div.bar')

// Getting a NodeList by class attribute
var myClassElements =
document.getElementsByClassName('bar')
// Getting a NodeList by element type
var myTags = document.getElementsbyTagName("li")
// Getting a NodeList by CSS Selector
var myClassElements =
document.querySelectorAll(".bar")
```

- Accessing individual nodes
  - GetElementById
  - QuerySelector
- Accessing lists of nodes
  - GetElementsByClassName
  - GetElementsByTagName
  - QuerySelectorAll
- Return an *Element* object or list of elements
  - Represents an individual HTML element
- Element objects have a TON of useful properties and methods
  - Getting information
  - Setting information

Traversing the DOM

- Element objects have a series of properties for accessing adjacent nodes
- These let you move around the DOM tree from any particular point



Useful node properties

```
// Grab a single element via id attribute
var myElement = document.getElementById('id')
// Navigate from that element
var myNewElement = myElement.parentNode // Move up the tree
var myNewElement = myElement.childNodes // Get list of children
var myNewElement = myElement.childNodes[number] // Select specific child
var myNewElement = myElement.firstChild // Get first child
var myNewElement = myElement.nextSibling // Move "right"
var myNewElement = myElement.previousSibling // Move "left"
```

Working with Elements

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

- Work with Text Nodes
  - **nodeValue**
  - **nodeName**
  - **nodeType**
- Work with HTML content
  - innerHTML
  - textContent
  - createElement
  - createTextNode
  - appendChild / removeChild
- Working with Attributes
  - className / id
  - hasAttribute
  - getAttribute
  - setAttribute
  - removeAttribute

Working with Elements

```
<div id="d1">…</div>
```

- `var myElement = document.getElementById("d1");`
- `myElement.nodeName // "DIV"`
- `myElement.nodeValue //null (unless text node)`
- `myElement.nodeType //1`

Working with Elements

```
// display HTML content
myElement.innerHTML;

// select text of 4th child
myElement.childNodes[3].textContent;

var para = document.createElement("p");
var text = document.createTextNode("Hi");
para.appendChild(text);
para.setAttribute("class", "quote");
para.className = "quote"
myElement.appendChild(para)
```

- Work with Text Nodes
  - nodeValue
  - nodeName
  - nodeType
- Work with HTML content
  - **innerHTML**
  - **textContent**
  - **createElement**
  - **createTextNode**
  - **appendChild / removeChild**
- Working with Attributes
  - className / id
  - hasAttribute
  - getAttribute
  - setAttribute
  - removeAttribute

Working with a NodeList

```
/* working with node lists */
var myNodelist = document.querySelectorAll("p");
var i;
for (i = 0; i < myNodelist.length; i++) {
  myNodelist[i].style.backgroundColor = "red";
}

for (i of myNodelist) {
  i.style.backgroundColor = "red" // i is the object, not an index
}
```

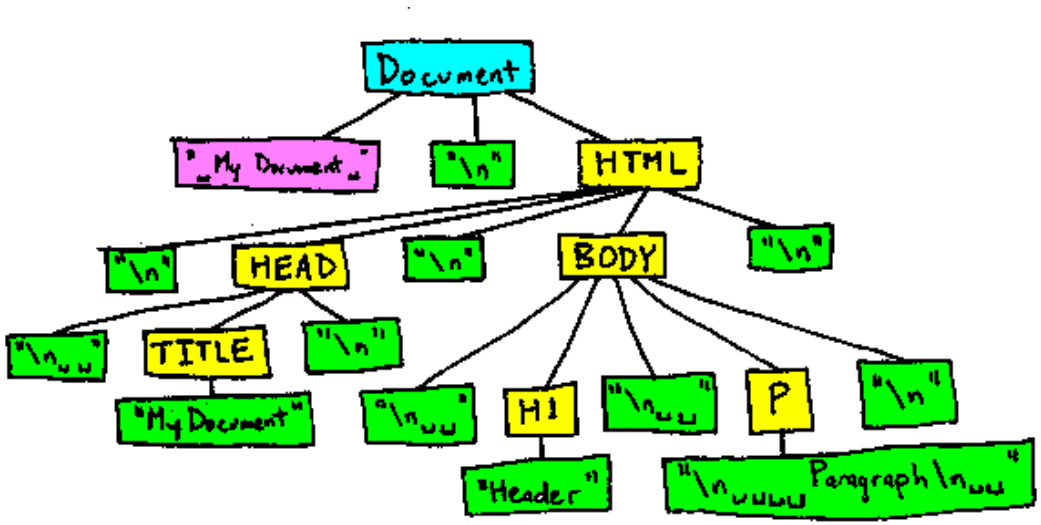
Storing DOM queries

```
// store all the paragraph elements
in a variable
const paragraphs =
document.getElementsByTagName("p")


// Select just the paragraphs for
those with a fancy class
const fancyParagraphs =
paragraphs.querySelectorAll(".fancy")
```

- You should save your DOM queries in a variable if you will be accessing them repeatedly
- Faster than DOM lookup
  - Not searching the whole tree
- Stores a pointer to the element(s)
  - LIVE

White Space - In the DOM

```
<!-- My document -->
<html>
<head>
  <title>My Document</title>
</head>
<body>
  <h1>Header</h1>
  <p>
    Paragraph
  </p>
</body>
</html>
```



Updating Text Nodes

```
const paragraphs =
document.getElementsByTagName("p")

for (p of paragraphs) {
  // Transform all the text
  let text = p.textContent
  text = text.toUpperCase()
  p.textContent = text
}
// All paragraph text is now uppercase
// don't do this at home
document.body.textContent =
document.body.textContent.toUpperCase()
```

- nodeValue
  - Only works with the *actual* text node, not the containing element
- textContent
  - returns all of the values of the text node in current element and its children
- innerText
  - similar, but wont get texts with CSS set to “display:none”
  - is slower because it accounts for layout rules

DOM Events

DOM Events

Event	Action
change	When an element has changed
click	Press and release mouse over element
keypress	When a character is inserted
mouseover	When mouse moves over element
load	When web page finishes loading
resize	When browser window size has been changed
scroll	When users scrolls up or down

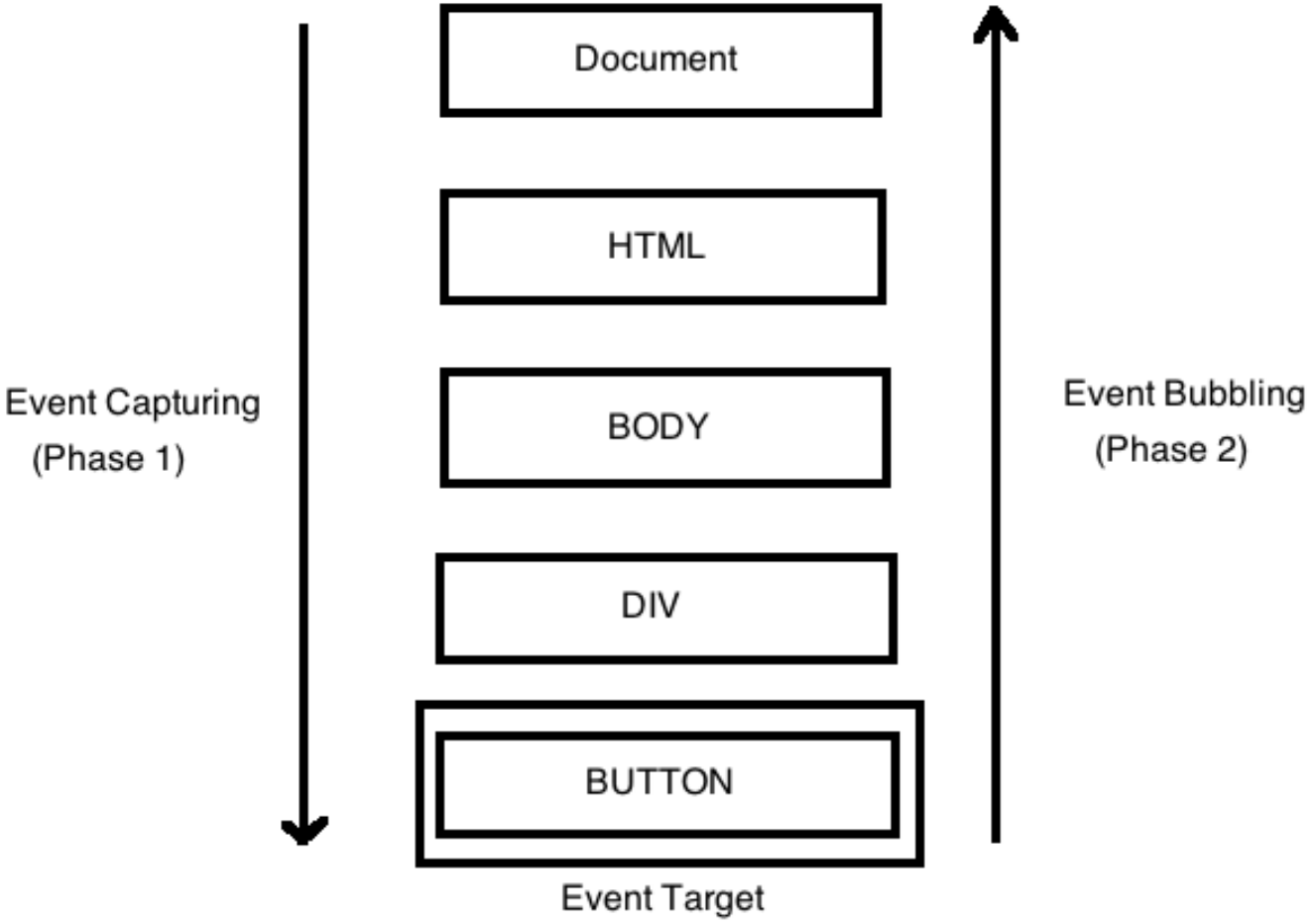
- Events allow you to execute JavaScript when the user has performed some action
- Events are **fired** or **raised** which then **triggers** a function or script
  - This allows you to add interactivity
  - Delay the execution of your JavaScript
- Process:
  - Select Element
  - Specify Event
  - Write a JS function that will execute
  - Bind JavaScript function to Element + Event
- TONS of events
  - MDN Event Reference

Binding Events to Elements

```
<script>
function myFunction() {
  Console.log("Paragraph changed.");
}
</script>

<button type="button" onclick="myFunction()">Try it</button>

// get header element
var header = document.getElementsByTagName('h1')[0];
// bind function
header.onclick = function(e){console.log("you clicked me!");}
//bind another function
header.onclick = function(e){console.log("Clicked Again!");}

//bind with event listeners
header.addEventListener("click", function(e){console.log("ok, that is enought");})
header.addEventListener("click", function(e){console.log("STAAAAAAP!!!!");})
```

- HTML Event Handlers
  - Don't use these
- DOM Event Handlers
  - Good support
  - Only one function per event
- DOM Event Listeners
  - Newest approach
  - Favored way
  - Multiple functions per event

Event Flow


Document
HTML
BODY
DIV
BUTTON
Event Capturing
(Phase 1)
Event Bubbling
(Phase 2)
Event Target

Inspector (Gadget) Demo

https://www.crockford.com/javascript/javascript.html

Assignment 1 - Description

- Hand-code a HTML, CSS, and JavaScript Website
  - Artisanal Websites
- Steps
  - Create a new "hello-webpage" project on Glitch
  - Edit the index.html, styles.css, and script.js to add your own HTML, CSS, and JavaScript.
  - Create new HTML pages for each page of your website
  - Tweak your HTML, CSS, and JavaScript and refresh the page
- Expectations - Make an entire website
  - 5 individual HTML pages, using a variety of elements
  - CSS that applies to the whole site, that modifies the layout and style of content
  - Javascript that adds interactivity
- Due: September 20th at 11:59pm

Activity 3 - Javascript & Interactivity

1. Go to https://glitch.com/~infsci2560-2023-activity3 and REMIX IT
2. Complete the FOUR tasks
    a. Look at README.md and the HTML Source for instructions
3. Submit the Glitch *project* URL
    a. Project URL - https://glitch.com/~infsci2560-2023-activity3 << **THIS**
    b. Live URL - https://infsci2560-2023-activity3.glitch.me/ << NOT THIS