Learning JavaScript

Edward Sharick - Week 6, 7 and 8 (10/2/23, 10/9/23, and 10/16/23)

What is JavaScript?

- JavaScript is a programming language that integrates with html elements and CSS rules.
- It can find HTML elements and modify their style properties (CSS) or their content.
- It can add more user-interactivity to your website, hide and show elements, etc.

Adding JavaScript to your webpage

- In an HTML file:
    - add a <script> tag then call the script when some event occurs (i.e. onclick)
    - scripts can be placed in the <head> and/or <body> section(s)
    - placing scripts at the bottom of the <body> can improve display speed, since script interpretation is slow
- External JavaScript:
    - An external JS file (someScript.js) can contain java script functions
    - Then in the HTML file, reference the filename with the 'src' property of the <script>
    - Use a local file path:

        <script src="/js/script1.js"></script>

        - Same folder – script.js
        - Folder inside of current folder – folderName/script.js
        - Path from root directory - /somepath/script.js
        - Path from one level up from current folder - ../somepath/script.js
    - Or use the full url:

        <script src="https://www.somesite.com/js/script1.js"></script>

    - External scripts are useful when you have many different web pages referencing the same code.
    - External scripts also separate HTML and code, which can make it easier to read and maintain.
    - External JS files can be cached by the browser, improving page loads.

Programming with JavaScript

Output of a program:

- You can output data from a script in different ways:
    - Write data into an HTML element (innerHTML)
    - Write data into the HTML output (document.write()) – puts the output where the script is in the head or body
        - Warning: after HTML document is loaded, this will delete all existing HTML

    - Write into an alert box (windows.alert())
        - You can just say 'alert()'
    - Write into a browser console (console.log())
- You can also print the contents of the current window with 'window.print()'

JavaScript Syntax

Java Script is a programming language, so it has the same common elements as other programming languages (variables, strings, selection, iteration, arrays, functions, objects, etc.). So just learn the syntax for these, the conceptual mechanisms will be the same for the most part.

Best Practices

- Minimize the use of global variables.
- Always declare and initialize local variables.
- Place local variables at the top of the script.
- Declare objects, arrays and Date with 'const'
- Don't use the 'new' keyword except for creating objects from classes.
- Use === comparison instead of ==
- Use parameter defaults – function(a=1, b=2){
- Avoid using eval()

Improve Speed of JS code execution

- Reduce activity in loops
- Reduce DOM activity
    - Save calls in variables: const x = document.getElementById("demo");
- Reduce DOM size
- Avoid unnecessary variables
- Load scripts at the bottom of the page

Statements

- Use ';' at the end of each executable statement (like java and C#)

Code Blocks

- Use curly brackets { .. } to define code blocks
- Use two spaces (not tab – browsers interpret tabs differently) for indents

Some reserved keywords

- var, let, const, if, switch, for, function, return, try

Comments

- // for single line or /* */ for block comment

Variables

- Use camelCase
- var, let and const are keywords for variables

```
var x = 5;
let z = 10;
const pi = 3.14;
```

- Declaring a variable but not initializing it gives it the 'undefined' value.
- JavaScript variables declarations are 'hoisted' – they are moved to the top of their scope
    - But 'const' and 'let' variables are not initialized, so can throw reference errors
    - "use strict"; //put at the beginning of the script; you cannot use undeclared variables

Var

- 'var' existed pre 2015; 'let' and 'const' added after 2015
- 'var' should only be used in code written for old browsers, or when you need a Global Scope variable
- 'var' variables can be redclared

Let

- 'let' variables have 'block scope; they shadow variables of the same name in their ancestors
- 'let' variables cannot be redeclared

Const

- 'const' variables cannot be changed (the reference to the value cannot be changed)
- You cannot reassign the variable, but you can change the elements of a constant array or the properties of a constant object
- Use 'const' if you know the value should not be changed
    - A new array, object, function or regexp

Operators

- Arithmetic operators:
    - +, -, *, /, %, ++, --, ** (exponentiation)
- Comparison operators:
    - ==, , !=, >, <, >=, <=
    - ? (ternary operator)
    - === (equal value and equal type)
    - !== (not equal value or not equal type)
    - Can be applied to strings and numbers
- Logical operators:
    - &&, ||, and !
- Type operators:
    - typeof variable – returns the type of the variable

    - can be “string”, “number”, “boolean”, “object”, “function” or “undefined”
        - The data type of NaN is number
        - The data type of an array is object
        - The data type of a date is object
        - The data type of null is object
        - The data type of an undefined variable is undefined *
        - The data type of a variable that has not been assigned a value is also undefined
    - If you want to check if an object is an Array or Date, use ‘x === Array’ or ‘x === Date’
    - instanceof(object) – returns true if an object is an instance of an object type
        - cars instanceof Array
- Void operator
    - void(expression) – evaluates the expression and returns undefined.

```
<a href="javascript:void(0);">
  Useless link
</a>
```

- Bitwise operators:
    - &, |, ~ (not), ^, <<, >>, >>> (unsigned right shift)
- Assignment operators:
    - =
    - (arithmetic) +=, -=, *=, /=, %=, **=
    - (logical) &&=, ||=
    - (null coalescing) x ??= 5; (if ‘x’ is undefined will be set to 5)
    - (bitwise) &=, |=, ^=, <<=, >>=, >>>=

Data Types

JS types are dynamic; there are 8 datatypes:

- String
- Number
    - stored as 64-bit floating point decimal
    - can use e-5 for exponential notation
    - don’t need to write the decimal point
- BigInt
    - Can be used to store large integers that are too big for normal JS numbers
    - let x = BigInt(“123451234123413245123413251325123 5”)
- Boolean
    - ‘true’ or ‘false’
- Undefined
    - ‘undefined’ – means a variable hasn’t been initialized, or has been initialized to undefined (let x = undefined;)
- Null
    - Null also means a variable has value ‘undefined’, but it is still type of ‘object

    - So null == undefined is true, but null === undefined is false.
- Symbol
- Object
    - The object data type can contain:
        - 1. An object
            - const person = {name: “John”, age:50 };
        - 2. An array
            - const vals = [“a”, “b”, “c”];
            - 0-based index
        - 3. A date

Type Conversions:

- Number()
- parseFloat()
- parseInt()
- toExponential()
- toFixed()
- toPrecision()
- String()
- toString()
- Some odd automatic conversions for JS:
    - “0” to Boolean -> true
    - “000” to Boolean -> true
    - “” to Number -> 0; to Boolean -> false
    - [] to Number -> 0
    - [20] to Number -> 20
    - [20, 30] to Number -> NaN
    - Null to number -> 0

Functions

- A block of code to be executed when called; can have parameters and return some value

```
function functionName(param1, param2){
        //function code
        return param1 + param2;
}
```

- Can use the => function (like lamda expressions)

```
let f = (a, b) => a * b;
```

- Self invoking functions

```
(function () {
  let x = "Hello!!";  // I will invoke myself
})();
```

- arguments.length – number of arguments; called within a function

- function.toString() – returns text of function
- function(…args) – like c# params modifier, it allows for any number of parameters
    - ‘arguments’ keyword is an array of the arguments when the function was invoked
- .call(), .apply(), and .bind() are all ways to call functions with different arguments

Callbacks, Async, Promises and Async/Await

- Callback functions are just functions passed as a parameter to another function that will be called within the body of that function.
- Callbacks can be used with asynchronous functions such as
    - setTimeout(function, time) //function is executed after time (in millis)
    - setInterval(function, time) //function is executed at each time interval
- Since callbacks can be difficult to debug and program, JS introduced ‘promises’
- A Promise contains both producing code (code that may take some time) and consuming code (code that must wait for the result of the producing code)
-

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

- A promise can be ‘Pending’, ‘Fulfilled’, or ‘Rejected’
- Async and Await make Promises easier to write:
    - Async – makes a function return a Promise
    - Await – makes a function await a Promise
        - The await keyword can only be used inside an async function.
        - The await keyword makes the function pause the execution and wait for a resolved promise before it continues.

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    setTimeout(function() {resolve("I love You !!");}, 3000);
  });
  document.getElementById("demo").innerHTML = await myPromise;
}
```

```
myDisplay();
```

Objects

- A container for name:value pairs; syntax is like a python dictionary
- You can access object properties by:
  - objectName.propertyName
  - objectName[“propertyName”]
- Objects can also have methods, stored as nameless functions:
  - fullName : function() { return this.firstName + " " + this.lastName; }
- Methods can be invoked with ():
  - objectName.fullName()
- ‘this’ keyword refers to the object
  - ‘this’ keyword has different meaning based on scope and when it’s used
  - Can use ‘call()’, ‘apply()’ and ‘bind()’ to change what object ‘this’ refers to in a function
- It is good practice to create a constructor function for objects, essentially it’s a workaround for not using classes.
  - The .prototype allows you to add a method or property to an existing constructor

Classes

- The class keyword is used to create a class; should have a constructor and use this to create fields:

```
class CarName {
        constructor(a){
                this.a = a;
        }
        method1(){
        }
        method2(){
        }
        …
}
```

- Then you can create an object:

```
let x = new CarName(“a”); //x.a = “a”;
```

- Can use the ‘get’ and ‘set’ keywords to create properties to get/set values
- Can create static methods
- Inheritance is supported with the keyword ‘extends’
  - Use ‘super’ to call the parent class’s constructor

Events

- HTML events are ‘things’ that happen to HTML elements and JS can ‘react’ to these events

    - Button clicked, input text changed, page finished loading, etc.
- Some event keywords:
    - onchange
    - onclick
    - onmouseover
    - onmouseout
    - onkeydown
    - onload
    - and more… (https://www.w3schools.com/jsref/dom_obj_event.asp)

Strings

- Used for storing and manipulating text
- Single quotes ‘ ‘, double quotes “ “ can be used
- .length – built-in property of a string variable
- Escape characters: \\, \’, \”, \n, \r, \t, \v, \b, \f
- Many built-in functions.
    - slice(start, end), substring(start,end), substr(start, count)
        - slice and substr support negative index
        - substring doesn’t
    - replace(), replaceAll(), toUpperCase(), toLowerCase(), concat(),
    - trim(), trimStart(), trimEnd()
        - get rid of whitespace
    - padStart(3, “x”), padEnd()
        - add something to the beginning or end until string has requested length
    - charAt(), charCodeAt()
        - returns index of character or charCode value at a position
    - split()
        - string to an array
    - searching stuff:
        - indexOf(), lastIndexOf() – can also take a 2nd parameter for starting position
        - search() – similar to index of but cannot take a 2nd param and can use powerful reg expressions
        - match(), matchAll(), includes(), startsWith(), endsWith() - can use strings or regexpressions to find matches in a string
- They pretty much act like python strings
- Template literals:

```
let text = `Welcome ${firstName}, ${lastName}!`;

//Act like the fstring in python (but you need to use the ` char not ‘)
```

    - They can also be used to create html elements inside code:

```
let header = "Templates Literals";
let tags = ["template literals", "javascript", "es6"];

let html = `<h2>${header}</h2><ul>`;
for (const x of tags) {
  html += `<li>${x}</li>`;
}

html += `</ul>`;
```

Numbers

- Integers are accurate up to 15 digits
    - Largest value is Number.MAX_SAFE_INTEGER; (and smallest is MIN)
    - use BigInt for larger numbers:
        - add 'n' to the end of the number 123n
        - or use BigInt(123)
    - BigInt cannot have decimals
    - Arithmetic operations between Number and BigInt isn't allowed
- Max number of decimals is 17
- Floating point arithmetic can have overflow error
- It will automatically try to convert strings to numbers for mathematic operations (except for "+" since that will use concatenation)
    - If it cannot be converted, it will do the calculation and return NaN
    - isNaN(x) returns true if x is NaN
- Infintiy and -Infinity are keywords
    - Number.POSITIVE_INFINITY, Number.NEGATIVE_INFINITY, and Number.NaN can all be used as Number properties
- You can represent a number as:
    - hex: 0xFF
    - binary: 0b1100;
    - octal: 0o432;
- Can convert to a string x.toString()
- Can set to a certain number of decimal places: x.toFixed(3); x.toExponential(3); x.toPrecision(2);
- You can parse input to a number:
    - Number("1")
    - parseInt("12 safd")
    - parseFloat("10.33")

Array

- Arrays are Objects, typeof returns "Object"
- Should be defined as const:
    - const vals = [1, 2, 3];

- Accessed by index (negative indices do NOT work)
- Some function for accessing elements and modifying elements the array:
    - .length – number of elements
    - Add elements with .push() – end - and .unshift() - beginning
    - Remove with .pop() – end – and .shift() – beginning
    - 'delete fruits[0]' – leaves 'undefined' at spot 0
    - .concat() – like pythons append() and extend() functions; can work on an element and an array
    - .splice(2, 0, "A", "B", "C", …); - starting at position 2, delete 0 elements and insert "A", "B", and "C"
        - Can be used to remove sections without leaving holes
    - slice(start, end) – gets a sublist from start to end (non-inclusive)
    - .reverse()
- Max and min
    - No built-in functions, but can use Math.max.apply(null, someArray);
    - More common to write a custom min/max
- Array iterations:
    - Can be traversed with a for loop
    - Can use the Arrays.forEach()
        - vals.forEach(someFunction());
- Other functions that iterate the array:
    - .sort() – sorts the array in place
        - Works for strings
        - Doesn't work for numbers, have to pass it a comparer function: fun(a, b) { return b-a;}
    - .flat() – flattens an array, reducing its dimensionality
    - …someArray – array spread, opposite of flat()
    - .map(someFunction), where someFunction has 3 parameters: f(value, index, array)
        - .flatMap() is similar, but applies the flat() function after
    - .filter(someFunction), only items that match the filter will be returned
    - .reduce(someFunction) to reduce the list to one element
    - .every(someFunction) return true if all match condition
    - .some(someFunction) return true if at least one matches condition
    - .indexOf(item, start), .lastIndexOf(item, start)
    - .find(someFunction) returns value of first item that passes condition
    - .findIndex(someFunction)
    - .includes(someValue) – returns true or false
    -
- Misc functions:
    - Array.from("ABCDEF") – makes array from string
    - .keys() – returns an iterator of keys (indices)

- .entries() – returns an iterator of key/value paris

Dates

- Created using the new Date() constructor – 9 different constructors
  - Specifying values over the limit will for day, month, second, hour, minute will overflow into the next one (i.e. 25 hours, will set to 1 hour and add a day)
- Dates are object type
- Dates can be parsed from certain string inputs
- Methods:
  - Getter methods: getFullYear(), getMonth(), getHours(), etc.
    - Mostly numeric – month is (0-11)
    - Helpful to use an array of the strings, then use the index
  - Date.now() – current time in milliseconds from 1/1/1970
  - Setter methods: setFullYear(), setMonth(), setHours(), etc.
- Dates can be compared

Math

- Some constants: Math.PI, Math.E, Math.SQRT2, Math.SQRT1_2, Math.LN2, Math.LN10
- Some functions: Math.round(), ceil(), floor(), trunc(), sign(), pow(), sqrt(), abs(), sin(), cos(), min(), max(), log(), log2(), log10(), etc.
- Math.random(); //returns a number [0, 1]
  - This function will return a random number including min and max:

```
function getRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1) ) + min;
}
```

Booleans

- Everything with a value is true; without a value (and 0) is false;

Conditionals: If, else, and switch

- JS uses if, else if, else pattern (same as java and c#)
- JS uses switch/case syntax (same as java and c#)

```
switch(expression) {
  case x:
     //code block
    break;
  default:
    //code
}
```

- Switch statements use strict comparison '===', which means it has to match value and type

Loops – For loops, while loops and iterables

- Same syntax as Java for traditional for loop:

```
for (let i = 0; i < cars.length; i++) {
```

```
  text += cars[i] + "<br>";
}
```

- For each loops similar, use ‘in’ instead of “:”:

```
for (key in object) {
  // code block to be executed
}
```

- For arrays, ‘in’ gives the index, not the value; use ‘of’ for this

```
for (car of carArray){
}
```

- While loops and do/while loops have the same syntax as Java
- ‘continue’ – next iteration
- ‘break’ – end iteration
- JavaScript allows you to use ‘break’ and ‘continue’ in any code blocks if you label the code block

```
someLabel: {
    //some code
    //break someLabel;
   //this code won’t be reached
}
```

Other collections – Sets and Maps

- Set: unique collection of elements
  - new Set(); //empty constructor
  - new Set([‘a’, ‘b’]); //existing collection
  - .size – property
  - .add() – add element to set
  - .delete()
  - .has()
  - .forEach()
  - .values()
- Maps: hold key: value pairs
  - new Map();
  - set(); //set value for a key
  - get(); //get value for a key
  - delete(); //delete based on key
  - has(); //checks based on key
  - forEach(function); //applies function to each key/value pair
  - entries(); //iterators with [key, value] pairs
  - size – property representing number of key, value pairs

Regular Expressions

- A sequence of characters that forms a search pattern

- Syntax
  - /pattern/modifiers;
- Patterns
  - [abc] – any character in brackets
  - [0-9]
  - [a-z]
- Metacharacter
  - \d – digit
  - \s – whitespace
  - \b – beginning of word
- Quantifiers
  - n+ - one or more
  - n* - zero or more
  - n? – zero or one
- Modifiers
  - i – case insenstitive
  - g – global (find all matches)
  - m – multi-line matching
- pattern.text(expression) //returns true or false
- pattern.exec(expression) //returns found text or null
- JavaScript RegExp Reference (w3schools.com)

Error handling

- Try, catch, finally blocks – same as Java
- Throw – same as Java
- Some errors:
  - Eval Error
  - RangeError
  - ReferenceError
  - SyntaxError
  - TypeError
  - URIError
  - And more… (https://www.w3schools.com/jsref/jsref_obj_error.asp)

Using code from other modules

- Like python, uses the 'import' keyword, and the type="module" (type can be omitted)

```
<script type="module">
import message from "./message.js";
</script>
```

- But the file has to include an 'export' keyword, where it exports named variables or functions.

JSON

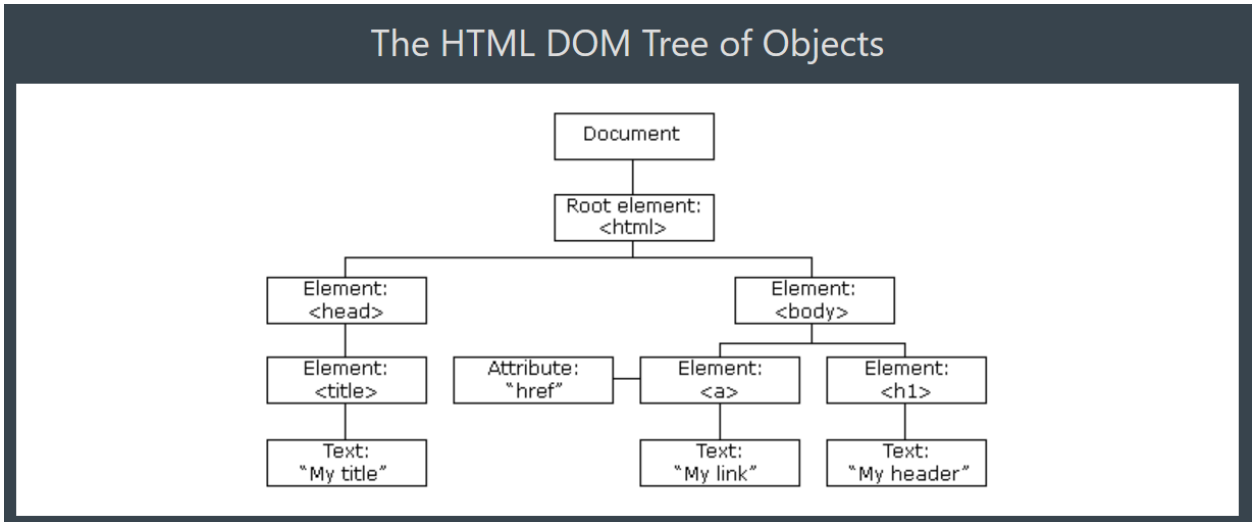
- JavaScript Object Notation – format for storing and transporting data as a long string
- JSON Syntax Rules:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays
- Example:

```
"employees":[
 {"firstName":"John", "lastName":"Doe"},
 {"firstName":"Anna", "lastName":"Smith"},
 {"firstName":"Peter", "lastName":"Jones"}
]
```

  - Employees is an array containing three objects
- Built-in JS functions to parse JSON
  - JSON.parse(text) – takes JSON text and returns the JS object
  - JSON.stringify(obj) – takes JS obj and returns a JSON string

JS HTML DOM

- HTML Document Object Model – a tree of objects based on the structure of the HTML page



- The DOM is a standard object model and programming interface for HTML, defining:
  - The HTML elements as objects
  - The properties of all HTML elements
  - The methods to access all HTML elements
  - The events for all HTML elements
- Methods:
  - 'getELementById("someID") – returns the HTML element as an object
    - If not found, it returns null

- ‘getElementsByTagName(“tagName”) – returns the HTML elements that match the tag, (p, a, div, etc.); returns them as an object HTML collection
- An HTML collection is like an array in terms of accessing elements, but you cannot use array methods (push(), pop(), etc.)
- ‘getElementsByClassName(“className”) – same as ‘tagName’ but with ‘className’
- ‘querySelectorAll(“p.intro”) – list of all <p> elements with class = “intro”
- ‘forms[“frm1”]’ – find a form element with the ‘id=”frm1”’
  - JS is often used for form validation
  - Setting an input to have the ‘required’ attribute can also auto validate forms

- Properties:
  - ‘innerHTML’ – change the innerHTML text
  - ‘attribute’ – change the attribute value
  - ‘style.property’ – change the CSS property
    - document.getElementById("p2").style.color = "blue";
    - <button type="button"
      onclick="document.getElementById('id1').style.color = 'red'">
      Click Me!</button>
  - List of Style object properties:
    - https://www.w3schools.com/jsref/dom_obj_style.asp
- Adding and Deleting elements from the DOM
  - document.createElement(element)
  - document.removeChild (element)
  - document.appendChild (element)
  - document.replaceChild (new, old)
  - document.write(text) – writes to HTML output stream
- Using the DOM for animations:
  - By updating the style properties gradually on time intervals, you can create animations with JS code.
  - Ex:

```
function myMove() {
  let id = null;
  const elem = document.getElementById("animate");
  let pos = 0;
  clearInterval(id);
  id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
```

```
        elem.style.top = pos + 'px';
        elem.style.left = pos + 'px';
      }
    }
  }
```

- Events and Event Listeners

  Mouse events
  - onclicked //when the element is clicked
  - onmouseover
  - onmouseout
  - wheel

  Key events
  - keydown
  - keypress
  - keyup

  Page events
  - onload //when the element loads
  - onunload //when the user leaves the page; useful for dealing with cookies

  Input/Form events
  - oninput //for input fields, when the input changes
  - onchange //when you leave an input field

  More events here: [HTML DOM Event Object (w3schools.com)]

  Event Listeners
  - You can add event listeners to any elements via the syntax:

    ```
    element.addEventListenere(event, function, useCapture);
    ```
  - You can add multiple event listeners to an elements, which can allow for dynamic events (events causing other events, etc.)
  - Bubbling vs event capturing
    - If you have a <p> inside a and the user clicks on the <p> which event should be handled first? The onclick of <p> or of ?
    - In bubbling, the inner most event is handled first: <p> before
    - In capturing, the outermost event is handled first: before <p>
    - The default value is bubbling (so useCapture is false)
- DOM Navigation
  - You can use different key words to access properties between node relationships:
    - parentNode
    - childNodes[nodenumber]
    - firstChild
    - lastChild
    - nextSibling
    - previousSibling

    - Then use .nodeValue to get the value
    - Special nodes:
        - document.body - The body of the document
        - document.documentElement - The full document
    - You can create node elements:
        - const para = document.createElement("p");
    - You can use 'appendChild(newElement)' to add it as a child to an existing node.
    - You can use 'insertBefore(newElement, otherChild)' to add it before another child
    - You can use 'removeChild' or 'remove' to remove an existing node
    - You can use 'replaceChild' to replace an existing node with another node

JS Browser Object Model (BOM)

- Allow access to the browsers window, screen, location, history, navigator, popup alerts, timing and cookies.
- Window – refers to the current browser's window:
    - window.innerHeight/Width
    - window.open(), close(), moveTo(), resizeTo()
- Screen – refers to the visitor's screen
    - screen.width/height
    - screen.availWidth/availHeight – width/height of screen minus interface features, like Windows taskbar
    - screen.colorDepth/pixelDepth
- Location – refers to url path location of the current browser's page
    - location.href
    - location.hostname
    - location.pathname
    - location.protocol
    - location.assign() – changes the location to a new page
- History – browser's history
    - history.back()
    - history.forward()
- Navigator
    - cookieEnabled – returns true or false
    - platform – operating system
    - language
- Alerts
    - alert("some text")
    - confirm("some text") – returns the input value if "OK" is clicked, or null if "Cancel" is clicked
    - prompt("some text", "defaultText") – returns the input value if "OK" or null if "Cancel"
- Timing

- setTimeout(function, time)
- setInterval(function, time)
- clearTimeout()
- clearInterval()

- Cookies
  - document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
  - useful to write functions to get, set and check cookie values

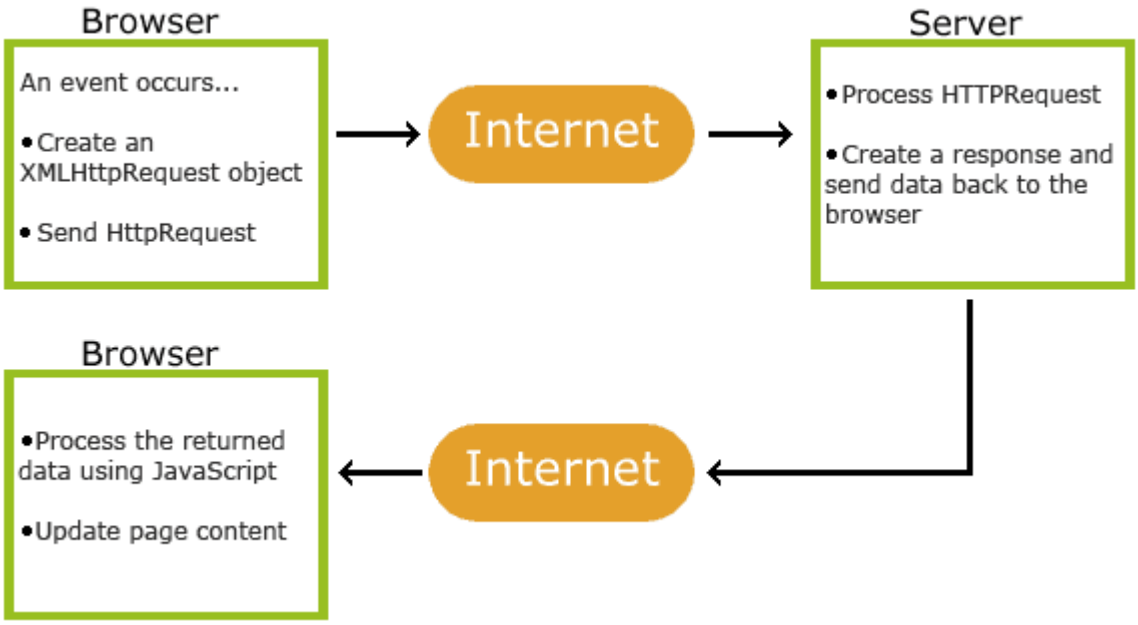
```
function setCookie(cname, cvalue, exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays*24*60*60*1000));
  let expires = "expires="+ d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}

function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i <ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}

function checkCookie() {
  let username = getCookie("username");
  if (username != "") {
   alert("Welcome again " + username);
  } else {
    username = prompt("Please enter your name:", "");
    if (username != "" && username != null) {
      setCookie("username", username, 365);
    }
  }
}
```

JS AJAX

- AJAX = Asynchronous JavaScript And XML.
- AJAX is not a programming language.
- AJAX just uses a combination of:
    - A browser built-in XMLHttpRequest object (to request data from a web server)
    - JavaScript and HTML DOM (to display or use the data)



- Can be used to get data from an XML or JSON file, send requests to a PHP or ASP file which can run a server side script or access a server side database.

```
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
};
xmlhttp.open("GET", "json_demo.txt");
xmlhttp.send();
```

- I may have to learn this more when I learn PHP in a few weeks. (JSON PHP (w3schools.com))

We can use JSON to send/receive data from server instead, which might be easier and better in some cases.

- Sending object data to a PHP file:

```
const myObj = {name: "John", age: 31, city: "New York"};
const myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

- Getting object data from a PHP file:

```
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "demo_file.php");
xmlhttp.send();
```

JS Graphics

- There are some JS libraries for different plots and charts:
  - Plotly.js – Plotly.js (w3schools.com)
  - Chart.js - Chart.js (w3schools.com)
  - Google Chart - Google Chart (w3schools.com)
  - D3.js - D3.js (w3schools.com)