JavaScript useful stuff

## Constant

* An initialized variable that cannot be changed
* Must be initialized when declared
* Ex. const testVariable = 8;

## prompt()

* The prompt function prompts the user with a dialog box that allows the user to type a single lin of text and press OK or Cancel
* Returns the string the user typed or null

## console.log()

* Displays text or numbers in the console

## Arithmetic with numbers and strings

* if you add a variable that is a string like “4” with a variable that is a number like 4 the number variable will be converted to a string
* “4” + 4 = “44”
* if you do any non addition arithmetic to a string and a number/string variable, the string variable is converted to a number
* “4” \* 4 = 16

## parseInt()

* converts strings into numbers
* if given a non-number as an argument, returns NaN

## parseFloat()

* converts strings into float numbers
* if given a non-number as an argument, returns NaN

## isNaN()

* returns true if variable is not a number

## ===

* is only true if both values are of the same variable type

## <

* When comparing strings it compares the Unicode value of the letters
* “a” < “b”
* “asdfewvavsdsdf” < “basdfef”
* lowercase letters have higher values than uppercase

## Conditional(Ternary) operator

* **condition ? expression1: expression2**
* if the condition evaluates to true, then the value of expression1 is returned otherwise, the value of espression2 is returned

## Switch statement

* uses the === when determining cases
* can have a default case

## Continue

* makes the loop instantly iterate again

## Naming conventions

### function names

* + - Verb followed by a noun
    - ex. displayAverage
  + use camel case

## Function

* Function will still run if you don’t give it all the arguments it is supposed to have

### function expression

* + - let exampleFunction = function(x, y, z) { console.log(x + y + x);}
    - need to declare the variable before using the function

### Arrow function

* + - let exampleFunction = (parameter1, parameter2) => parameter1 + parameter2
    - you can combine arrow functions with Ternary operator\
    - if the function takes one parameter then no need for ()
    - no need for return statement, the function automatically returns whatever is in the body

njn

## Array

| Method | Description | Example |
| --- | --- | --- |
| push(value) | Adds a value to the end of the array | let nums = [2, 4, 6];  nums.push(8); *// nums = [2, 4, 6, 8]* |
| pop() | Removes the last array element and returns the element | let nums = [2, 4, 6];  let x = nums.pop(); *// returns 6, nums = [2, 4]* |
| unshift(value) | Adds a value to the beginning of the array | let nums = [2, 4, 6];  nums.unshift(0); *// nums = [0, 2, 4, 6]* |
| shift() | Removes the first array element and returns the element | let nums = [2, 4, 6];  let x = nums.shift(); *// returns 2, nums = [4, 6]* |
| splice(startingIndex,  totalElementsToDelete,  valuesToAdd) | Adds or removes elements from anywhere in the array and returns the deleted elements (if any) | let nums = [2, 4, 6, 8, 10];  *// Deletes all elements from index 3 to the end*  nums.splice(3); *// nums = [2, 4, 6]*  *// Deletes 2 elements starting at index 0*  nums.splice(0, 2); *// nums = [6]*  *// Adds 3, 5 starting at index 0*  nums.splice(0, 0, 3, 5); *// nums = [3, 5, 6]*  *// Adds 7, 9, 11 starting at index 2*  nums.splice(2, 0, 7, 9, 11); *// nums = [3, 5, 7, 9, 11, 6]* |

* for-of loops
  + loops through an array
  + for(variable of array) {}
* forEach()
  + loops through an array
  + array.forEach(function(item, index) {}
* indexOf()
  + search beginning of array to end and returns index of item or -1
* lastIndexOf()
  + search end of array to beginning and returns index of item or -1
* sort()
  + default sorts alphabetically/unicode value

## Objects

* is an unordered collection of properties
* an object property is a name-value pair, where the name is a string and the value is any data type
* properties and methods are separated by commas
* object literal is a comma-separated list of property name and value pairs
* object is created by
  + let object = {}
* object methods
  + access object’s properties using the keywaord *this*
  + ex. this.someProperty
  + can be created inside or outside an object
    - ex. someMethod: function(){}
    - ex. object.someMethod: function(){}
* getter
  + is a function that is called when an object’s property is retrieved
  + ex. get property() { return someValue; }
* setter
  + is a function that is called when an object’s property is set to a value
  + ex. set property(value)
  + called by
    - object.property = value
* accessor property
  + is an object property that has a getter or setter or both
  + is not a method does not need () when being called
* for-in loop
  + iterates over an object’s properties in arbitrary order

## 0Map

* a type of object that store key/value pairs which are known as elements
* created with
  + let variable = new Map()
* object.keys
  + returns an array of an object’s property names
* in operator
  + returns true if an object contains the given property anreturns false otherwise
  + ex. key in object
* delete operator
  + removes a key/property from map or object
  + ex. delete object[key]
* set(key, value) method
  + method sets a key/value pair. If the key is new, a new element is added to the map. If the key already exists, the new value replaces the existing value.
* get(key) method
  + gets a key's associated value
* has(key) method
  + returns true if a map contains a key, false otherwise
* delete(key)
  + method removes a map element.
* size property
  + is the number of elements in the map.
* use for of loop to iterate through

## String

* charAt()
  + returns character at specified index
  + someString.charAt(3) == e
    - same as someString[3] == e
* length
  + someString.length
* indexOf()
  + method returns the index of the search string's first occurrence inside the String object or -1 if the search string is not found.
* lastIndexOf()
  + returns the index of the search string’s last occurrence inside the string object or -1 if the search string is not found
* replace()
  + replaces one string with another and returns the string with the replacement string inside

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| substr() | Returns the substring that begins at a given index and has an optional given length. | s = "As you wish.";  s.substr(3, 3); *// "you"*  s.substr(3); *// "you wish." (length optional)* |
| substring() | Returns the substring between two indices, not including the second index. | s = "As you wish.";  s.substring(3, 6); *// "you"*  s.substring(3); *// "you wish." (2nd index optional)* |
| split() | Returns an array of strings formed by splitting the string into substrings. The given delimiter separates substrings. | s = "As you wish.";  s.split(" "); *// ["As", "you", "wish."]* |
| toLowerCase() | Returns the string converted to lowercase characters. | s = "What?";  s.toLowerCase(); *// "what?"* |
| toUpperCase() | Returns the string converted to uppercase characters. | s = "What?";  s.toUpperCase(); *// "WHAT?"* |
| trim() | Returns the string with leading and trailing whitespace removed. | s = " test ";  s.trim(); *// "test"* |

* Template literal
  + replaces string concatenation
  + `${expression}

## Date

* Represents a single moment in time based on the number of milliseconds since the Unix Epoch(January 1, 1970)
* Created with the *new* operator and a constructor
  + A constructor is a function that creates an instance of an object
  + new Date() == current date and time
* Can input a specific date you want
* Months are 0-11
  + January == 0
  + February == 1

| * Method | * Description | * Example |
| --- | --- | --- |
| * getDate() setDate() | * Gets or sets the day relative to the current set month | * let day = new Date(2016, 0, 30); * day.getDate(); *// 30* * day.setDate(21); *// 30 -> 21* |
| * getDay() | * Returns the day of the week (0-6) | * let day = new Date(2016, 0, 30); * day.getDay(); *// 6 = Saturday* |
| * getFullYear() setFullYear() | * Gets or sets the 4 digit year | * let day = new Date(2016, 0, 30); * day.getFullYear(); *// 2016* * day.setFullYear(2017); *// 2016 -> 2017* |
| * getHours() setHours() | * Gets or sets the hour (0-23) | * let day = new Date(2016, 0, 30, 5, 0); * day.getHours(); *// 5* * day.setHours(2); *// 5 -> 2* |
| * getMilliseconds() setMilliseconds() | * Gets or sets the milliseconds (0-999) | * let day = new Date(2016, 0, 1, 5, 20, 10, 250); * day.getMilliseconds(); *// 250* * day.setMilliseconds(500); *// 250 -> 500* |
| * getMinutes() setMinutes() | * Gets or sets the minutes (0-59) | * let day = new Date(2016, 0, 30, 5, 20); * day.getMinutes(); *// 20* * day.setMinutes(35); *// 20 -> 35* |
| * getMonth() setMonth() | * Gets or sets the month (0-11) | * let day = new Date(2016, 0, 30, 5, 20); * day.getMonth(); *// 0* * day.setMonth(3); *// 0 (Jan) -> 3 (Apr)* |
| * getSeconds() setSeconds() | * Gets or sets the seconds (0-59) | * let day = new Date(2016, 0, 1, 5, 20, 10, 250); * day.getSeconds(); *// 10* * day.setSeconds(45); *// 10 -> 45* |
| * getTime() setTime() | * Gets or sets the number of milliseconds since Jan 1, 1970, 00:00:00 UTC | * let day = new Date(2016, 0, 30, 5, 20); * day.getTime(); *// 1454152800000* * day.setTime(1454153700000); *// Sat Jan 30 2016 05:35:00 GMT-0600* |

## Math

* Math.PI
  + Value of PI, approximately 3.142
* Math.E
  + Euler’s constant, approximately 2.18
* Math.LN2
  + Natural logarithm of 2,approximately 0.693
* Math.LOG10E
  + Base 10 logarithm of E, approximately 0.434
* Math.SQRT2
  + Square root of 2, approximately 1.414

|  |  |  |
| --- | --- | --- |
| * Method | * Description | * Example |
| * abs(x) | * Returns the absolute value of x | * Math.abs(-5); *// 5* |
| * ceil(x) | * Returns x rounded up to the nearest integer | * Math.ceil(2.1); *// 3* |
| * cos(x) | * Returns the cosine of the radians x | * Math.cos(Math.PI) *// -1* |
| * floor(x) | * Returns x rounded down to the nearest integer | * Math.floor(2.9) *// 2* |
| * log(x) | * Returns the natural logarithm of x | * Math.log(Math.E) *// 1* |
| * max(n1, n2, n3, ...) | * Returns the largest number | * Math.max(5, 2, 8, 1) *// 8* |
| * min(n1, n2, n3, ...) | * Returns the smallest number | * Math.min(5, 2, 8, 1) *// 1* |
| * pow(x, y) | * Returns x to the power of y | * Math.pow(2, 3) *// 8* |
| * round(x) | * Returns x rounded to the nearest integer | * Math.round(3.5) *// 4* |
| * sin(x) | * Returns the sine of radians x | * Math.sin(Math.PI) *// 0* |
| * sqrt(x) | * Returns the square root of x | * Math.sqrt(25) *// 5* |
| * tan(x) |  |  |

## Exception handling

* Is the process of catching and responding to an exception
* Exception
  + is an error that disrupts the normal flow of program execution
* Throw
  + throws a user-defined exception
* Try-catch
  + has a try block to execute code that may throw an exception and a catch block that executes when an exception is thrown
* Finally
  + Executes whether an exception was thrown or not

## toFixed()

* number.tofixed(n) returns a number with n decimal places

## Error

* Error object represents a runtime error, which is an error that occurs when the program is executing
  + name – The error’s name
  + message – The error’s message
* Error constructor takes a message as a parameter
  + new Error(“My error message”)

## Strict mode

* makes a JavaScript interpreter apply a set of restrictive syntax rules to JavaScript code
* To enable strict mode for an entire script, the statement "use strict" must be placed before any other statements.
  + can be enabled in only a function by placing “use strict” at the beginning of the function

## DOM

* Document Object Model
  + a data structure that represents all parts of an HTML document
* Web pages add JavaScript code by using <script> tag
* The DOM is accessible via the global object named document.
* document.getElementById()
  + returns the DOM node whose id attribute is the same as the method's parameter.
  + Ex: document.getElementById("early\_languages") returns the p node in the HTML below.
* document.getElementsByTagName()
  + returns an array of all the DOM nodes whose type is the same as the method's parameter.
  + Ex: document.getElementsByTagName("li") returns an array containing the four li nodes from in the HTML below.
* document.getElementsByClassName()
  + returns an array containing all the DOM nodes whose class attribute matches the method's parameter.
  + Ex: document.getElementsByClassName("traditional") returns an array containing the ol node with the class attribute matching the word traditional.
* document.querySelectorAll()
  + returns an array containing all the DOM nodes that match the CSS selector passed as the method's parameter.
  + Ex: document.querySelectorAll("li a") returns an array containing the two anchor nodes in the HTML below.
* document.querySelector()
  + returns the first element found in the DOM that matches the CSS selector passed as the method's parameter.
  + querySelector() expects the same types of parameters as querySelectorAll() but only returns the first element found while navigating the DOM tree in a depth-first traversal.
  + Ex: document.querySelector("li") returns the li node about Fortran.
* Modifying DOM node attributes
  + Change which image is displayed by modifying an img element's src attribute.
  + Determine which image is currently displayed by reading the img element's src attribute.
  + Change an element's CSS styling by modifying an element's style attribute.
* Every attribute for an HTML element has an identically named property in the element's DOM node
  + Ex. <a href="https://www.nasa.gov/" id="nasa\_link">NASA</a> has a corresponding DOM node with properties named href and id. Each attribute property name acts as both a getter and a setter.
  + Getter: Using the property name to read the value allows a program to examine the attribute's value
    - Ex. nasaUrl = document.getElementById("nasa\_link").href assigns nasaUrl the string "https://www.nasa.gov/" from the anchor element's href attribute
  + Setter: Writing to a property allows a program to modify the attribute, which is reflected in the rendered web page
    - Ex. document.getElementById("nasa\_link").href = "https://www.spacex.com/" changes the element's hyperlink to the SpaceX URL
* setAttribute
  + set a specific attribute
  + element.style.backgroundColor = "red";
* removeAttribute()
  + remove an elements attribute
  + Ex: document.getElementById("nasa\_link").removeAttribute("href") removes the link from the anchor element so that clicking on the HTML element no longer performs an action.
* Modifying DOM node content
  + textContent
    - gets or sets a DOM node's text content
    - Ex: document.querySelector("p").textContent = "$25.99"; changes the paragraph to <p>$25.99</p>
  + innerHTML
    - gets or sets a DOM node's content, including all of the node's children, using an HTML-formatted string
    - Ex: document.querySelector("p").innerHTML = "<strong>$25.99</strong>"; changes the paragraph to <p><strong>$25.99</strong></p>
* Hidden attribute
  + is not displayed by the web browser
* document.documentElement
  + is the root of the DOM tree
  + Ex. let html = document.documentElement; assigns the html variable with the HTML document's root node
* parentNode
  + property refers to the node's parent
* childNodes
  + property is an array-like collection of objects for each of the node's children
* children
  + property is similar to the childNodes except the array contains only element nodes and no text nodes
* nextElementSibling
  + property refers to the element node with the same parent following the current node in the document
* previousElementSibling
  + property refers to the element node with the same parent preceding the current node in the document
* Modifying the DOM structure
  + appendChild()
    - appends a DOM node to the object's child nodes
  + insertBefore()
    - inserts a DOM node as a child node before an object's existing child node
  + removeChild()
    - removes a node from the object's children
    - The most common usage pattern is to get a DOM node, n, and call removeChild() on n's parent passing n as an argument.
      * Ex: n.parentNode.removeChild(n)
  + createElement()
    - returns a new element node, created from a string argument that names the HTML element
    - Ex: document.createElement("p") creates a new paragraph node
  + createTextNode()
    - returns a new text node containing the text specified by a string argument
    - Ex: document.createTextNode("Hello there!") creates the text node with "Hello there!", which can then be appended to an element node.
  + node method cloneNode()
    - returns an identical copy of a DOM node
    - The method's boolean argument indicates whether the method should also clone the node's children
      * Ex. x.cloneNode(true) creates an identical tree rooted at x
      * Ex. x.cloneNode(false) creates only a single node identical to x
  + After creating or cloning a node, the node does not appear in the web page until the node is attached to the DOM tree. A programmer must use appendChild() or insertBefore() to add the new node to the existing DOM tree

## writeln()

* outputs HTML into the document and alters the DOM
* ex. document.writeln("<div>test</div>")
* writeln is spelled with a lowercase L

## Window object

* Represents an open browser window
* in a tabbed browser each tab has a window object
* The document object is a property of the window object and can be accessed as window.document or just document this is the same with all window properties
* *window.location*
  + is a location object that contains information about the window's current URL.
  + Ex: window.location.hostname is the URL's hostname.
* *window.navigator*
  + is a navigator object that contains information about the browser.
  + Ex: window.navigator.userAgent returns the browser's user agent string.
* *window.innerHeight* and *window.innerWidth* 
  + are the height and width in pixels of the window's content area.
  + Ex: window.innerWidth returns 600 if the browser's content area is 600 pixels wide.
* *window.alert()*
  + displays an alert dialog box.
    - Ex: window.alert("Hello") displays a dialog box with the message "Hello".
* *window.confirm()*
  + displays a confirmation dialog box with OK and Cancel buttons. confirm() returns true if OK is pressed and false if Cancel is pressed.
  + Ex: window.confirm("Are you sure?") displays a dialog box with the question.
* *window.open()* opens a new browser window. Ex: window.open("http://www.twitter.com/") opens a new browser that loads the Twitter web page.

## Console object

* console.log()
  + displays informational data to the console.
* console.warn()
  + displays warnings to the console. The browser usually has a special indicator to differentiate a warning from the standard log message.
  + Ex: A yellow warning box.
* console.error()
  + displays errors to the console. The browser usually has a special indicator to differentiate an error from a warning or the standard log message.
  + Ex: A red error box.
* console.dir()
  + displays a JavaScript object to the console. The browser usually supports a method for compactly representing the object.
  + Ex: a hierarchical tree representation allowing a developer to expand and collapse the object contents.

## Script

* use <script> tags to load JavaScript from external files rather than writing the JavaScript directly within the HTML file. The <script> tag's src attribute specifies a JavaScript file to load
* ex. <script src="bootstrap.js"></script>
* async attribute
  + allows the browser to process the web page concurrently with loading and processing the Javascript
* defer attribute
  + allows the browser to load the web page concurrently with loading the JavaScript, but the JavaScript is not processed until the web page is completely loaded

## CSSOM

* CSS Object Model
* a set of APIs that allow JavaScript to manipulate CSS properties of a web page
* getPropertyValue()
  + returns the value of an element's CSS property or an empty string if the property is not set
  + Ex: elem.style.getPropertyValue("color") gets the element's color value.
* setProperty()
  + sets the value of an element's CSS property
  + Ex: elem.style.setProperty("color", "blue") sets the element's color to blue
* removeProperty()
  + removes an element's CSS property
  + Ex: elem.style.removeProperty("color") removes the element's color property
* The style CSS properties can alternatively be accessed and modified using JavaScript property names
  + Ex: elem.style.color = "blue" is equivalent to elem.style.setProperty("color", "blue").
* CSS property names that have dashes are converted into property names that use camel case.
  + Ex: background-color becomes the JavaScript property backgroundColor.
* Modifying a stylesheet
  + The document.styleSheets object is a list of the stylesheets used in the web page
  + Each stylesheet in document.styleSheets is a CSSStyleSheet object, which maintains a list of the stylesheet's CSS rules in the property called cssRules
  + insertRule()
    - inserts a new rule into the stylesheet
    - Ex: document.styleSheets[0].insertRule("p { color: blue; }") inserts a new paragraph rule that makes the text color blue.
  + deleteRule()
    - deletes a rule at a given index number from the stylesheet
    - Ex: document.styleSheets[0].deleteRule(0) deletes the first CSS rule from the first stylesheet
* a rule's CSS properties can be accessed, modified, and removed with getPropertyValue(), setProperty(), and removeProperty()
  + Ex: document.styleSheets[0].cssRules[0].style.setProperty("color", "blue") sets the stylesheet's first rule's color to blue
* classList properties
  + Every DOM node has a classList property that lists the classes assigned to the node
    - Ex: The div node created from <div class="account warning"> has a classList with items "account" and "warning".
  + add()
    - adds a class to the node's classList
    - Ex: elem.classList.add("mystyle") adds the class mystyle to the element's list of classes
  + remove()
    - removes a class from the node's classList. Ex: elem.classList.remove("mystyle") removes the class mystyle from the element's list of classes
  + toggle()
    - method adds the class to the node's classList if the class is not present.
    - If the class is already present, the class is removed
    - Ex: elem.classList.toggle("mystyle") toggles the class mystyle on or off
  + className property
    - modifies the node’s class list directly, which is a space-delimited list of the classes assigned to the node
      * Ex: elem.className = "cat adopted" assigns the cat and adopted classes to the element and removes any previously assigned classes from the node
    - All classes assigned to className are also added to the node's classList

## setTimeout()

* Executes a function after a time delay
* Takes two arguments
  + A function
  + A time delay in milliseconds(1/1000th of a second)
* The browser calls the function after the time delay
* Returns a unique integer identifier that refers to the timeout
* Can be cancelled by passing the identifier to clearTimerout()

## setInterval()

* Execute a function repeatedly with a time delay between calls
* Takes two arguments
  + A function
  + A time interval in millliseconds
* The browser calls the function every *t* milliseconds until the interval is cancelled
* Returns a unique integer identifier that refers to the timeout
* Can be cancelled by passing the identifier to clearInterval()

## Web Storage API

* provides storage objects that allow JavaScript programs to securely store key/value pairs in the web browser
* Web Storage API supports two storage objects
  + sessionStorage
    - stores key/value pairs for an origin that are only available for the duration fo the session
    - Closing the browser or browser tab ends the session
  + localStorage
    - stores key/value pairs for an origin that are stored indefinitely
* origin
  + a combination of scheme, hostname, and portnumber in a URL
  + Ex.
    - http://example.com/
    - http://www.example.com/
    - https://www.example.com/
    - <http://www.example.com:8080>
  + The browser stores the data for each origin separately and does not share the data between origins.
* Local and session storage methods, ex. localStorage.setItem(testKey, testValue)
  + setItem(key, value)
    - stores the key string and associated value string in storage
  + getItem(key)
    - returns the value associated with the key in storage or null if the key does not exist
  + removeItem(key)
    - removes the key and associated value from storage
  + clear()
    - removes all keys and associated values from storage

## JSON

* an efficient, structured format for data based on a subset of the javaScript language
* has six basic data types
  + String
  + Number
  + Object
  + Array
  + Boolean
  + Null
* The JSON structure is defined recursively so that objects can contain arrays and arrays can contain objects to any arbitrary depth.

## JSON object

* JSON.parse()
  + creates a JavaScript object from a string containing JSON
  + Typically, JSON.parse() is used with data received from a server.
  + Ex: JSON.parse('[1,"two",null]') converts the string '[1,"two",null]'into the JavaScript array [1,"two",null]
  + reviver function
    - optional second parameter of .parse()
    - modify parsed values before being returned
    - can buy used when a JSON string represents a data type not available in JSON
      * Ex: A reviver function can convert a string representing a date, "2010-12-30", to a JavaScript Date object.
* JSON.stringify()
  + creates a string representation of any passed object by either calling the object's toJSON() method if defined or recursively serializing all enumerable, non-function properties
  + Typically, JSON.stringify() is used with data sent to a server
  + Ex: JSON.stringify(new Date('2020-08-06')) converts the JavaScript Date object to the string 2020-08-06T00:00:00.000Z by calling the Date object's toJSON() method.
  + replacer
    - Enables customization of the generated string
    - if replacer is a function, JSON.stringify() will use the value returned by the function as the string representation
    - If replacer is an array, JSON.stringify() will filter the returned value by converting only the properties listed in the replacer array
      * Ex: JSON.stringify({a:1,b:2,c:3},["a","b"]) returns the string '{"a":1,"b":2}'
  + spacer
    - controls the indentation spacing of output JSON string, which indicates the depth of values in the object
    - When the spacer parameter is specified and not an empty string, the output will also include newlines
    - Ex: JSON.stringify({a:1,b:2}, null, " ")

## Ajax

* is a technique to asynchronously communicate with a server and update a web page once the response is received, without reloading the whole web page
* occurs when the web application sends a request to the server and continues running without waiting for the server response

## XMLHttpRequest

* an object for communicating with web servers using Ajax
* allows web browsers to hide the communication latency and continue to provide a responsive user interface while waiting for a server response
* defines handlers for events that occur during the request/response cycle
* The steps for using the XMLHttpRequest API are:
  + Create a new XMLHttpRequest object.
  + Assign handlers to the desired events via the addEventListener() method.
    - The addEventListener() method takes two arguments: the event name and the event handler
    - code that should execute when the event occurs.
    - If the handlers are not set up prior to calling the open() method, the progress events will not execute.
  + Initialize a connection to a remote resource using the open() method
    - The open() method takes two arguments: the HTTP request type and the URL for the resource
    - Most browsers only support "GET" and "POST" request types
  + Modify the default HTTP request headers if needed with the setRequestHeader() method
    - x: xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded") sets the Content-Type header so a URL-encoded string may be sent in a POST request.
  + Send the HTTP request via the send() method
    - For POST requests, the data to be sent with the request is passed as the argument to the send() method
* Result handlers
  + load
    - is called when the exchange between browser and server has completed
    - is called after any progress handlers
  + error
    - is called when the browser does not receive an appropriate response to a request
  + abort
    - is called when the browser is told to stop a request/response that is still in progress
  + timeout
    - is called if the browser takes too much time to fully receive a response to a request
    - by default, the browser does not provide a timeout for a request
* Progress handlers
  + loadstart
    - is called when the browser begins to send a request
    - is called before any other handlers
  + loadend
    - is called after the browser receives the response
    - is called upon both response success and failure, and is called after all other handlers
  + progress
    - is called one or more times while a response is being received by the client
    - is called before result handlers
    - can be used to provide a data download progress indicator to the user. A similar handler is available to provide an indicator for uploaded data.
    - is passed an event object
* has attributes for checking the status of a response, which are usually used in the load handler and used to update the DOM.
  + Status
    - is the numeric status code returned in the response
    - ex. this.status === 401
  + StatusText
    - is the descriptive text describing the status attribute
  + The status code identifies the specific reason for a failure response
* The XMLHttpRequest object provides multiple ways to access the response data.
  + response attribute
    - is the response body, which is parsed by the browser according to the responseType attribute
    - Use this when you want to access the object of a xhr object
      * ex. this.response.property
  + responseText attribute
    - is the plain text version of the response
  + responseXML attribute
    - is the XML DOM version of the response.
    - is only available as a DOM object if the response is a valid and correctly formatted XML document
  + responseType attribute
    - is set by the programmer to let the browser know the expected response data format
    - if the responseType is set to “json”, then the browser parses the entire response as a JSON object and sets the response attribute to the JSON object
    - if the responseType attribute is “” or “text”, the browser leaves the response unprocessed, and the response attribute contains the same value as responseType
    - if the responseType attribute is “document”, the browser assumes the response is an XML document, and the response attribute contains the same value as responseXML
* upload
  + is an object for monitoring the status of the request being sent to the server.
  + The upload attribute has the same handlers as the XMLHttpRequest object, but the progress handler is the only handler typically used for the upload attribute.
  + The progress handler can be used to monitor the status of uploading large files, such as attaching a document to a Gmail message.
  + Must use a post request not a get
* Content-Type
  + The Content-Type is used by both web servers and browsers to understand the file type. A web server keeps track of the file type in case an uploaded file is requested by another browser.

## Third-party web APIs

* a public web API used by a web application to access data provided by a third party
* To use a third-party web API, a developer usually registers with the third party to obtain a API key, the key
  + identifies who or what application is using the web API
  + helps the third party limit the number of requests made to the API in a fixed time period or may be used to charge a developer a fee for additional requests
  + to obtain a key, developers must agree to restrictinos the third party places on data obtained from the web API
* RESTful web API
  + is a web API that is called with a URL that specifies API parameters and returns JSON or XML containing the API data
* Weather API
  + OpenWeatherMap provides a free Weather API providing current weather data, forecasts, and historical data. Developers must register at openweathermap.org for an API key that must be transmitted in all API requests.
  + The OpenWeatherMap website provides documentation explaining how to use the Weather API using GET requests with various query string parameters. The API endpoint http://api.openweathermap.org/data/2.5/weather returns the current weather based on the following query string parameters:
  + zip - Five digit US ZIP code
  + units - Standard, metric, or imperial units to use for measurements like temperature and wind speed
  + appid - Developer's API key
  + Other parameters are documented in the OpenWeatherMap website. The Weather API returns weather data in JSON format by default
  + Try 5.11.1: Try OpenWeatherMap's API in your web browser.
    - Go to openweathermap.org.
    - Sign up for an account to obtain an API key.
    - When your API key is ready, try the link: http://api.openweathermap.org/data/2.5/weather?zip=90210&units=imperial&appid=APIKEY to make an API request for the weather with ZIP 90210. The page should indicate an invalid API key was used.
    - Replace APIKEY in the URL's query string with your API key, and reload the web page. The JSON-encoded weather information for 90210 should be displayed.
    - Change the ZIP code in the URL's query string to your ZIP code, and reload the URL to see the weather in your ZIP code.
* Cross-origin requests
  + Calling a third-party web API from the web browser requires a cross-origin HTTP request, since the web API is not hosted on the local website's web server
  + Two main techniques are used to make cross-origin requests
    - Cross-Origin Resource Sharing (CORS)
      * is a W3C specification for how web browsers and web servers should communicate when making cross-origin requests.
      * requires the web browser to send an Origin header in a web API request to indicate the scheme and domain making the API request
      * If the API accepts the request, the API responds with an Access-Control-Allow-Origin header indicating the same value in the Origin request header or "\*", which indicates that requests are allowed from any origin
      * uses other headers that begin with Access-Control-\* to support other interactions with the API
      * allows the browser to send GET, POST, PUT, and DELETE requests
        + JSONP limits the browser to sending only GET requests.
    - JSON with Padding
      * is a technique to circumvent cross-origin restrictions by injecting <script> elements dynamically into a web page. Script elements have no cross-origin restrictions.

## Common status codes

|  |  |
| --- | --- |
| Status code | Meaning |
| 200 | HTTP request successful |
| 3XX | General form for request redirection errors |
| 301 | Resource permanently moved, the new URL is provided |
| 4XX | General form for client errors |
| 400 | Bad request. Ex: Incorrect request syntax |
| 401 | Unauthorized request. Ex: Not properly authenticated. |
| 403 | Request forbidden. Ex: User does not have necessary permissions. |
| 404 | Not found. Ex: Requested resource does not exist. |
| 5XX | General form for server error codes |
| 500 | Internal server error. Ex: Server-side code crashed. |
| 503 | Service unavailable. Ex: Web page is temporarily unavailable due to site maintenance. |

## RegExp()

* regular expression
* a string pattern that is matched against a string
* ex. let re = new RegExp(“abc”)
  + the pattern “abc” matches any string that contains “abc”
  + “abcde” matches
  + “abd” does not match
* .test() is used to test for whether the string matches or not

## Special characters

| Character | Description | Example |
| --- | --- | --- |
| \* | Match the preceding character 0 or more times. | /ab\*c/ matches "abc", "abbbbc", and "ac". |
| + | Match the preceding character 1 or more times. | /ab+c/ matches "abc" and "abbbbc" but not "ac". |
| ? | Match the preceding character 0 or 1 time. | /ab?c/ matches "abc" and "ac", but not "abbc". |
| ^ | Match at the beginning. | /^ab/ matches "abc" but not "cab". |
| $ | Match at the end. | /ab$/ matches "cab" but not "abc". |
| | | Match string on the left OR string on the right. | /ab|cd/ matches "abc" and "bcd". |

## Character ranges

* square brackets are used in regular expressions to match any character in a range of characters
  + ex. /[aeiou]/ matches any vowle
  + ex. /[0-9]/ matches any digit
* ^
  + negates a range
    - ex. [^str] matches any character except s, t, or r

## Metacharacters

* is a character or character sequence that matches a class of characters in a regular expression

| Metacharacter | Description | Example |
| --- | --- | --- |
| . | Match any single character except newline. | /a.b/ matches "aZb" and "a b". |
| \w | Match any word character (alphanumeric and underscore). | /a\wb/ matches "aAb" and "a5b" but not "a b". |
| \W | Match any non-word character. | /a\Wb/ matches "a-b" and "a b" but not "aZb". |
| \d | Match any digit. | /a\db/ matches "a2b" and "a9b", but not "aZb". |
| \D | Match any non-digit. | /a\Db/ matches "aZb" and "a b", but not "a2b". |
| \s | Match any whitespace character (space, tab, form feed, line feed). | /a\sb/ matches "a b" but not "a4b". |
| \S | Match any non-whitespace character. | /a\Sb/ matches "a!b" but not "a b". |

## Mode modifiers

* also called a flag
* changes how a regex matches and is placed after the second slash in a regex

|  |  |  |
| --- | --- | --- |
| Mode modifier | Description | Example |
| i | Case insensitivity - Pattern matches upper or lowercase. | /aBc/i matches "abc" and "AbC". |
| m | Multiline - Pattern with ^ and $ match beginning and end of any line in a multiline string. | /^ab/m matches the second line of "cab\nabc", and /ab$/m matches the first line. |
| g | Global search - Pattern is matched repeatedly instead of just once. | /ab/g matches "ab" twice in "cababc". |

## exec(str)

* determines what part of the string ‘str’ matches a regex
* returns a result array, or returns null if the pattern does not match

## Form validation

* data validation
  + checking input for correctness
  + .preventDefault()
    - cancels form
  + can use global variables to track whether the field is currently valid

## jQuery

* <script src=”jquery-3.5.1.min.js”></script>
  + or
    - <script src="https://code.jquery.com/jquery-3.5.1.min.js"
    - integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
    - crossorigin="anonymous"></script>
* jQuery function
  + jQuery() or $()
  + returns a jQuery object
    - contains a collection of DOM nodes
  + use variables that start with “$” to hold jQuery objects
  + takes a document.getElement() as an argument
* addClass()
  + is used to add a CSS class to selected elements as illustrated in the animation below
* removeClass()
  + is used to remove a CSS class from a selected element
* toggleClass()
  + is used to toggle a CSS class on and off
* Basic jQuery selectors
  + Element
    - $(“”)
  + ID
    - $(“#”)
  + Class
    - $(“.”)
* Chain classes together
  + $(“”).addclass().addclass()
* Additional selectors
  + Attribute selector - Selects elements based on an element attribute.
  + Basic filter selector - Selects elements based on a variety of properties.
  + Child filter selector - Selects child elements based on location or other properties.
  + Content filter selector - Selects elements based on an element's contents.
  + Hierarchy selector - Selects elements based on an element's location within the DOM hierarchy.

| Selector Type | Example | Explanation |
| --- | --- | --- |
| Attribute | $("span[id]") | Selects all <span> that have an id attribute |
| Attribute | $("a[href$='.pdf']") | Selects all <a> with href attributes ending in .pdf |
| Basic filter | $("p:first") | Selects the first <p> element |
| Basic filter | $("tr:even") | Selects the first, third, fifth, etc. table rows (zero-indexed) |
| Basic filter | $("li:eq(1)") | Selects the second <li> element (index n) |
| Child filter | $("li:last-child") | Selects the last <li> in each group |
| Content filter | $("p:contains('bye')") | Selects all <p> that contain the word "bye" |
| Hierarchy | $("li span") | Selects all <span> that are descendants of <li> |

* callback function
  + a function that is executed when an event occurs
  + called an event listener or event handler
  + on()

| Event | Shortcut | Description |
| --- | --- | --- |
| click | click() | Triggered when the mouse clicks on an element. |
| dblclick | dblclick() | Triggered when the mouse double-clicks on an element. |
| mouseover / mouseout | mouseover() / mouseout() | Triggered only once when the mouse pointer moves over or leaves an element and any of the element's children. |
| mouseenter / mouseleave | mouseenter() / mouseleave() hover() | Triggered only once when the mouse pointer moves over or leaves an element. hover() binds a callback function, or functions, for both events |
| mousemove | mousemove() | Triggered when the mouse pointer moves over an element. |

* ready event
  + is triggered when the browser has finished loading the web page’s DOM
  + $()

| Event | Shortcut | Description |
| --- | --- | --- |
| keydown | keydown() | Triggered when the user first presses a key on the keyboard. |
| keyup | keyup() | Triggered when the user releases a key from the keyboard. |
| keypress | keypress() | Triggered when the browser registers keyboard input from printable character keys. Non-printing keys like Shift and Esc do not register keypress events. |

| Event | Shortcut | Description |
| --- | --- | --- |
| focus / blur | focus() / blur() | Triggered when an element gains or loses focus. |
| focusin / focusout | focusin() / focusout() | Triggered when an element or any of the element's children gain or lose focus. |
| change | change() | Triggered when an element's value changes. |
| select | select() | Triggered when a user selects text in an <input type="text"> or <textarea> element. |
| submit | submit() | Triggered when the user is attempting to submit a form. |

* css() method
  + CSS properties can be added to selected elemenets with the css() method
    - ex. $(“body”).css(“background-color”, “peachpuff”)
  + callind css() with only a CSS property returns the current property value
* Animation methods

| Methods | Example | Description |
| --- | --- | --- |
| show()  hide()  toggle() | $("h1").show("slow");  $("h1").hide("slow");  $("h1").toggle("slow"); | Alters width, height, and opacity all at once |
| fadeIn()  fadeOut()  fadeToggle() | $("h1").fadeIn("normal");  $("h1").fadeOut("normal");  $("h1").fadeToggle("normal"); | Alters opacity only |
| slideDown()  slideUp()  slideToggle() | $("h1").slideDown("fast");  $("h1").slideUp("fast");  $("h1").slideToggle("fast"); | Alters height only |

* Speed argument for animation methods

| Argument | Example | Explanation |
| --- | --- | --- |
| "slow" | $("p").show("slow"); | 0.6 seconds to show the paragraph |
| "normal" | $("p").show("normal"); | 0.4 seconds to show the paragraph |
| "fast" | $("p").show("fast"); | 0.2 seconds to show the paragraph |
| milliseconds | $("p").show(1500); | 1.5 seconds to show the paragraph |

* queue()
  + aids in queuing code that should be executed after the previous animation complete
  + takes a function argument that is passed a function parameter called ‘next’
    - ‘next’ function must be called so the next animation in the queue can be processed
* animate()
* attr()
  + gets and sets attribute values of a DOM element
  + when attribute name and value is given, current value is replaced
  + when attribute name is given, returns current value
* The $() function creates new DOM nodes when given an HTML string
  + Ex: $("<span>I'm a new node!</span>")
* Methods for adding DOM nodes

| Methods | Example | Before | After |
| --- | --- | --- | --- |
| prependTo()  prepend() | $("<li>New first</li>").prependTo("ol");  *// same as*  $("ol").prepend("<li>New first</li>"); | <**ol**>  <**li**>A</**li**>  <**li**>B</**li**>  </**ol**> | <**ol**>  <**li**>New first</**li**>  <**li**>A</**li**>  <**li**>B</**li**>  </**ol**> |
| appendTo()  append() | $("<li>New last</li>").appendTo("ol");  *// same as*  $("ol").append("<li>New last</li>"); | <**ol**>  <**li**>A</**li**>  <**li**>B</**li**>  </**ol**> | <**ol**>  <**li**>A</**li**>  <**li**>B</**li**>  <**li**>New last</**li**>  </**ol**> |
| insertBefore()  before() | $("<p>Before</p>").insertBefore("h2");  *// same as*  $("h2").before("<p>Before</p>"); | <**h2**>Test</**h2**> | <**p**>Before</**p**>  <**h2**>Test</**h2**> |
| insertAfter()  after() | $("<p>After</p>").insertAfter("h2");  *// same as*  $("h2").after("<p>After</p>"); | <**h2**>Test</**h2**> | <**h2**>Test</**h2**>  <**p**>After</**p**> |
| wrap() | $("p").wrap("<div></div>"); | <**p**>A</**p**>  <**p**>B</**p**> | <**div**>  <**p**>A</**p**>  </**div**>  <**div**>  <**p**>B</**p**>  </**div**> |
| wrapAll() | $("p").wrapAll("<div></div>"); | <**p**>A</**p**>  <**p**>B</**p**> | <**div**>  <**p**>A</**p**>  <**p**>B</**p**>  </**div**> |
| wrapInner() | $("p").wrapInner("<div></div>"); | <**p**>A</**p**>  <**p**>B</**p**> | <**p**>  <**div**>A</**div**>  </**p**>  <**p**>  <**div**>B</**div**>  </**p**> |

* The jQuery methods remove() and detach() remove DOM nodes. Both methods are identical except detach() returns the removed nodes to the caller as a jQuery object in case the developer wants to use the nodes for other purposes.

| Methods | Example | Before | After |
| --- | --- | --- | --- |
| remove() | $("li").remove(); | <**ol**>  <**li**>A</**li**>  <**li**>B</**li**>  </**ol**> | <**ol**>  </**ol**> |
| detach() | let $listElems = $("li").detach(); | <**ol**>  <**li**>A</**li**>  <**li**>B</**li**>  </**ol**> | <**ol**>  </**ol**> |

* Methods for modifying DOM text

| Methods | Example | Before | After |
| --- | --- | --- | --- |
| html() | let s = $("p").html();  $("div").html(s); | <**p**>  A<**b**>B</**b**>C  </**p**>  <**div**>  </**div**> | <**p**>  A<**b**>B</**b**>C  </**p**>  <**div**>  A<**b**>B</**b**>C  </**div**> |
| text() | let s = $("p").text();  $("div").text(s); | <**p**>  A<**b**>B</**b**>C  </**p**>  <**div**>  </**div**> | <**p**>  A<**b**>B</**b**>C  </**p**>  <**div**>  ABC  </**div**> |

## Other