Javascript

Contents

- ECMA Script
  - Variables
  - Expressions
  - Output
  - Numbers
  - Strings
  - Arrays
  - Objects
  - Branching
  - Looping
  - Functions
  - Classes
- Forms API
- HTML DOM
- Browser DOM

Note: For every HTML element you need to know by memory, how to set and get values in JS and jQuery. Make a map

https://www.w3schools.com/jsref/

Contents

- Jquery
- AJAX
- JSON

Javascript Practice

- https://www.w3schools.com/js/js_exercises.asp

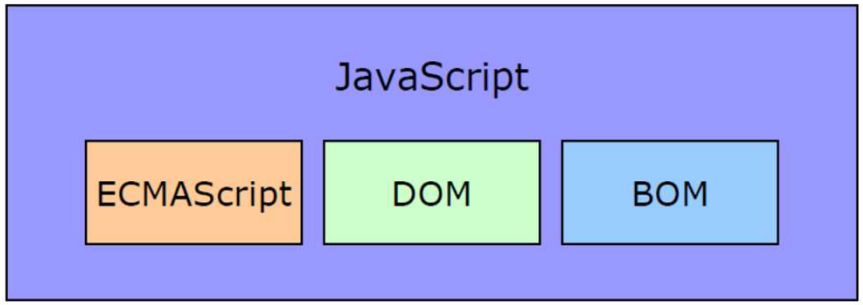
Javascript

- JavaScript is the programming language for HTML and the Web.
- Can be written with in the HTML code in <head> and <body>
- The best practice is to write it in a separate file.
- Placing Javascript in external files has the following advantages:
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Cached JavaScript files can speed up page loads
- Reference to external Javascript file

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

Javascript Implementaitons

- The complete Javascript implementations are made of three part



ECMA Script

- It is simple a description, defining all properties, methods and objects of a scripting language
  - Syntax
  - Types
  - Statements
  - Keywords
  - Reserved Words
  - Operators
  - Objects
- Each browser has its own implementation of the ECMA Script interface, which is then extended to contain DOM and BOM

DOM

- The DOM describes methods and interfaces for working with the content of the web page
- DOM is a tree based, language independent API for HTML and XML
- Document object is the only object that belongs to both DOM and BOM
- Some functions defined are:
  - `getElementsByTagName(), getElementsByName(), getElementById()`
- All attributes are included in HTML DOM elements as properties
  - `oImg.src = “picture.jpg”;`
  - `oDiv.className = “footer”;  // cf.class → className`

BOM

- The BOM describes methods and interfaces for interacting with the browser
- Each browser has its own implementations
- The window object represents the entire browser window:
  - Objects
    - Document : anchors, forms, images, links, location
    - Frames, history, navigator, screen
  - Methods
    - `moveBy(), moveTo(), resizeBy(), resizeTo()`
    - `open(), close(), alert(), confirm(), input()`
    - `setTimeOut(), clearTimeOut(), setInterval(), clearInterval()`
  - Properties
    - `screenX, screenY, status, defaultStatus, etc`

Outputs

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using **`innerHTML`**
  - Writing into the HTML output using **`document.write()`**
  - Writing into an alert box, using **`window.alert()`**
  - Writing into the browser console, using **`console.log()`**
  - The document can be sent to the printer using **`window.print()`**

Variables

- Variable are containers for storing data values
- They can be declared as below:

```
var x = 5;
var y = 6;
var z = x + y;

var person = "John Doe", carName = "Volvo", price = 200;
```

Primitive Types

- JavaScript has five primitive types:
  - Undefined
    - The Undefined type has only one value, *undefined*.
  - Null
    - The Null type has only one value, *null*.
  - Boolean
    - The Boolean type has two values, *true* and *false*.
  - Number
    - 32-bit integer and 64-bit floating-point values.
    - Infinity, isFinite()
    - NaN (Not a Number), isNaN()
  - String
    - Using either double quotes(”) or single quote(‘).
    - JavaScript has no character type.

Operators

- Unary
  - delete, void, Prefix ++/--, Postfix ++/--, Unary +/-
- Bitwise
  - ~, &, |, ^, <<, >>, >>>
- Boolean
  - !, &&, ||
- Arithmetic
  - +, -, *, /, %
- Assignment
  - =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=
- Comparison
  - ==, !=, >, >=, <, <=, ===, !==
- Conditional
  - variable = boolean_expression ? true_value : false_value;

typeof

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined" *
typeof null                   // Returns "object"
```

Strings

- Javascript strings are used for storing and manipulating text

```
var x = "Ram";
var y = new String("Ram");

// typeof x will return string
// typeof y will return object
// (x == y) is true because x and y have equal values
// (x === y) is false because x and y have different types (string and object)


str[0];

str[0] = ‘A’;  // Does not work
```

String Methods

```
var strln = txt.length;
var pos = str.indexOf("locate");
var pos = str.lastIndexOf("locate");
var pos = str.search("locate");
var res = str.slice(7, 13);
var res = str.substring(7, 13);
var res = str.substr(7, 6);
var n = str.replace("Old", "New");
var n = str.replace(/OLD/i,  "New");
var n = str.replace(/OLD/g,  "New");
```

Start point

```
var pos = str.indexOf("locate", 15);
```

Negative Indexing

```
var res = str.slice(-12, -6);
```

No Negative Indexing

Second parameter is the length of the extracted string

Regex, \i specifies case insensitive match

Regex, \g replace all occurances

String Methods

```
var text2 = text1.toUpperCase();

var text2 = text1.toLowerCase();

var text3 = text1.concat(" ", text2);     var text = "Hello".concat(" ", "World!");

var str = "       Hello World!        ";
alert(str.trim());

str.charAt(0);

str.charCodeAt(0);

txt.split(",");

words.join(",");
```

Numbers

- In Javascript, numbers can be written with or without decimals
- Numbers are always 64 bit floating point

```
var x = 3.14;      // A number with decimals
var y = 3;         // A number without decimals
var x = 123e5;     // 12300000
var y = 123e-5;    // 0.00123
var y = new Number(123);
```

- **NaN** is a JavaScript reserved word indicating that a number is not a legal number.
- **Infinity** (or **-Infinity**) is the value JavaScript will return if you calculate a number outside the largest possible number.

Numbers

```
var x = 10;
var y = 20;
var z = x + y; // z will be 30 (a number)

var x = 10;
var y = "20";
var z = x + y; // z will be 1020 (a string)

var x = "10";
var y = "20";
var z = x + y; // z will be 1020 (concatenates)
```

Number Methods

```
var x = 9.656;
x.toString();
x.toExponential(2);     // returns 9.66e+0
x.toFixed(0);           // returns 10
x.toPrecision(2);       // returns 9.7
x.valueOf();            // returns 123 from variable x
Number("10.33");        // returns 10.33
parseInt("10.33");      // returns 10
parseFloat("10 20 30"); // returns 10
parseFloat("years 10"); // returns NaN
```

Number Properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
POSITIVE_INFINITY	Represents infinity (returned on overflow)
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value

```
var x = Number.MAX_VALUE; // Returns the largest possible number

var x = 6;
var y = x.MAX_VALUE;    // y becomes undefined, cannot be used on variables
```

Arrays

- Arrays are used to store multiple values in a single variable

```
var cars = ["Saab", "Volvo", "BMW"];
var cars = new Array("Saab", "Volvo", "BMW");
```

- Elements in the array can be accessed through subscripting

```
var name = cars[0];
cars[0] = "Opel";
```

- Arrays are objects, Array elements can also be objects

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Methods

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];


var x = fruits.pop();      // the value of x is "Mango"

var x = fruits.push("Kiwi");   //  the value of x is 5

var x = fruits.shift();    // the value of x is "Banana", removes first element

var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5

delete fruits[0];  // Changes the first element in fruits to undefined

fruits.splice(2, 2, "Lemon", "Kiwi");

fruits.splice(0, 1); // Removes

fruits.splice(2, 0, "Lemon")  // Inserts at position 2
```

splice() can be used to add new items to the array
First parameter is position, Second is how many

Array Methods

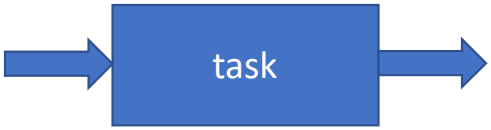
```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var arr4 = arr1.concat(arr2, arr3);   // Concatenates arr1 with arr2 and arr3

var arr5 = arr1.concat("Peter"); // Also accepts strings

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
var citrus = fruits.slice(1, 3);

var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();        // First sort the elements of fruits
fruits.reverse();     // Then reverse the order of the elements

var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
```

Numeric sort should be done using a compare function. It is passed as a parameter to the sort()

Array Iteration: foreach, map

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
```

Call a function for each array element




```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Creates a new array after applying the specified function for each array element

Array Iteration: filter, reduce

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

The filter() method creates a new array with array elements that passes a test.

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array)
{
  return total + value;
}
```

The reduce() method runs a function on each array element to produce (reduce it to) a single value.

Array Iteration: some, every

```
var numbers = [45, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

The some() method check if some array values pass a test.

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

The every() method check if all array values pass a test.

Array Iteration: find, findIndex

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.find(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

The find() method returns the value of the first array element that passes a test function.

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

The findIndex() method returns the index of the first array element that passes a test function.

Objects

- Objects are variables that can contain many values
- The values are written as name:value pairs

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- The named values, in JavaScript objects, are called **properties**.
- Methods are **actions** that can be performed on objects.

```
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Objects

- Objects are Mutable. They are addressed by reference, not by value.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}

var x = person;
x.age = 10;           // This will change both x.age and person.age
```

- Accessing Values

```
person.firstname + " is " + person.age + " years old.";

person["firstname"] + " is " + person["age"] + " years old.";
```

Objects

- Methods can also be written in the objects

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

In a function definition, this refers to the "owner" of the function.

```
name = person.fullName();
```

Objects

- Common solution to extract and use the values in the object

```
var person = {name:"John", age:30, city:"New York"};

document.getElementById("demo").innerHTML = person;
// Simply displays [object object], not of any use
document.getElementById("demo").innerHTML =
person.name + "," + person.age + "," + person.city;
var x, txt = "";

for (x in person) {
txt += person[x] + " ";
};

document.getElementById("demo").innerHTML = txt;
var myArray = Object.values(person);
```

Goes through all names

All values are extracted

Objects

```
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  language : "",
  get lang() {
    return this.language;
  }
  set lang(newlang) {
    this.language = newlang
  }
};

// Set an object property using a setter:
person.lang = "en";

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.lang;
```

Accessors

Objects

- It’s a good practice to use constructors

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

var p1 = new Person("John", "Doe", 50, "blue");
var p2 = new Person("Sally", "Rally", 48, "green");
```

Branching

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Looping

```
var cars = ['BMW', 'Volvo', 'Mini'];
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

```
var x;

for (x of cars) {
  document.write(x + "<br >");
}
```

```
var txt = 'JavaScript';
var x;

for (x of txt) {
  document.write(x + "<br >");
}
```

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
  text += person[x];
}
```

Looping

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

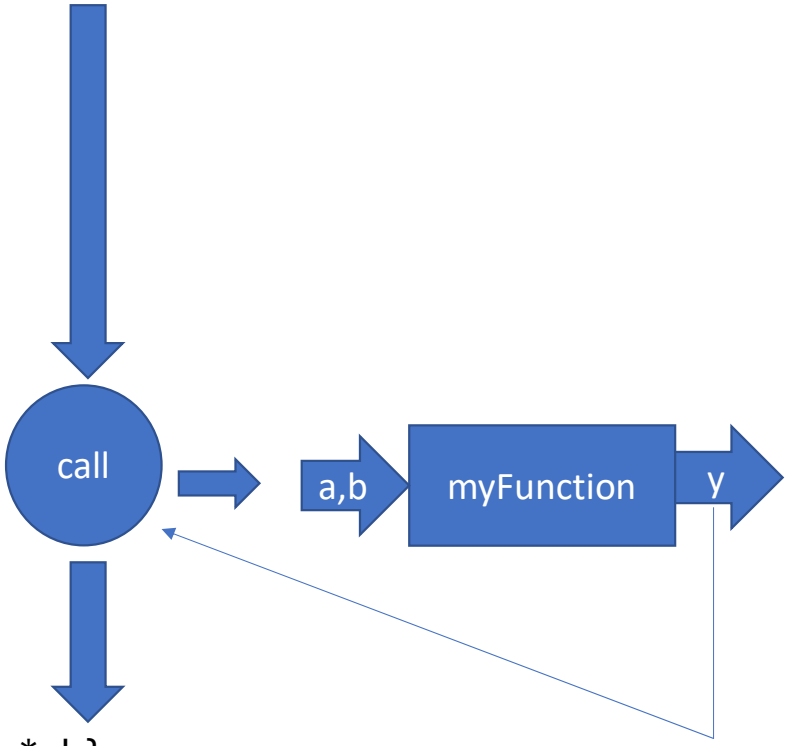
```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

Functions

- Functions are sub-programs
- Functions can be declared and defined

```
function myFunction(a, b) {
        y = a * b;
        return y;
}
```



- Function can also be used as expressions

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

- Functions can be created using function constructor

```
var myFunction = new Function("a", "b", "return a * b");
var x = myFunction(4, 3);
```

Functions

- Self invoking functions

```
(function () {
  var x = "Hello!!";  // Invoked Automatically
})();
```

- Arrow functions
  - Allows a short syntax for writing function expressions
  - They are not hoisted. They should be declared before they are used
  - They are not well suited for defining object methods

```
// ES5
var x = function(x, y) {
  return x * y;
}
```

```
// ES6
const x = (x, y) => x * y;
```

Functions

- Parameters can be passed into functions and default values can also be specified

Default value

```
function myFunction(x = 0, y) {
  if (y === undefined) {
    y = 0;
  }
}
```

Use undefined to check if the values missed while calling

Functions

- With the call() and apply() method, a method that can be used on different objects can be written

```
var person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
var person1 = { firstName:"John", lastName: "Doe" }
var person2 = { firstName:"Mary", lastName: "Doe" }

person.fullName.call(person1);  // Will return "John Doe"
person.fullName.call(person1, "Oslo", "Norway"); // Will add Oslo, Norway
person.fullName.apply(person2);  // Will return "Mary Doe“
person.fullName.apply(person1, ["Oslo", "Norway"]);
```

Classes

- A class is a type of function, but instead of using the keyword function to initiate it we use **class**
- The properties are assigned inside a **constructor()** method

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return "I have a " + this.carname;
  }
}

mycar = new Car("Ford");
document.getElementById("demo").innerHTML = mycar.present();
```

Class

- Getters and Setters

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

mycar = new Car("Ford");

document.getElementById("demo").innerHTML = mycar.cnam;
```

Class

- Static Methods

Static methods aren't called on instances of the class. Instead, they're called on the class itself. These are often utility functions, such as functions to create or clone objects.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  static hello(x) {
    return "Hello " + x.carname;
  }
}

mycar = new Car("Ford");

document.getElementById("demo").innerHTML = Car.hello(mycar);
```

Class name used

Class

- Inheritance

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

mycar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = mycar.show();
```

Forms API

- HTML form validation can be done by JavaScript

```
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

Function can be called when the form is submitted

```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

Constraint Validation in Forms

Property	Description
checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validationMessage property of an input element.

```
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
  var inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
  }
}
</script>
```

Constraint Validation in Forms

Property	Description
validity	Contains boolean properties related to the validity of an input element.
validationMessage	Contains the message a browser will display when the validity is false.
willValidate	Indicates if an input element will be validated.

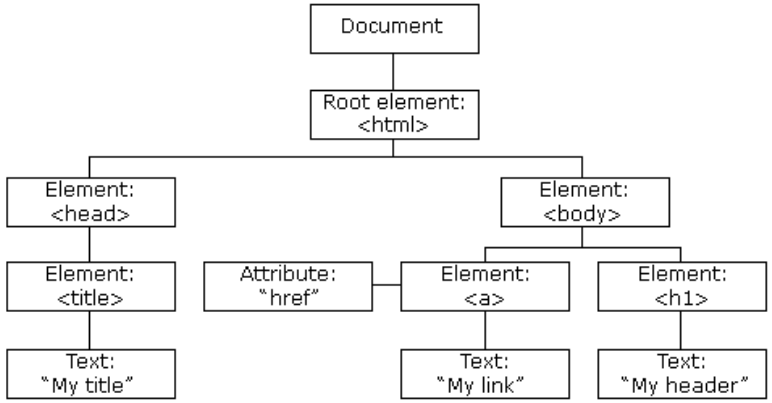
```
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
  var txt = "";
  if (document.getElementById("id1").validity.rangeOverflow) {
    txt = "Value too large";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
```

Validity Properties

Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

HTML DOM

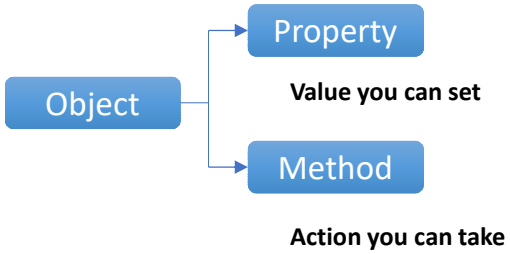
- The browser creates the Document Object Model when the webpage loads
- It is created as a tree of objects
- With the DOM, JS can do the following:
  - Change all the HTML elements in the page
  - Change all the HTML attributes in the page
  - Change all the CSS styles in the page
  - Remove existing HTML elements and attributes
  - Add new HTML elements and attributes
  - React to all existing HTML events in the page
  - Create new HTML events in the page



The HTML DOM is a standard for how to get, change, add, or delete HTML elements

HTML DOM

- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements




```
<html>
<body>

<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

Document Object

- Document Object represents the webpage
- All objects are accessed through the document object
- Use the `document` keyword to access the document object

https://www.w3schools.com/js/js_htmldom_document.asp

Finding HTML Elements

- To manipulate HTML elements, first it should be accessed
- Finding HTML elements by id

```
var myElement = document.getElementById("intro");
```

- Finding HTML elements by tag name

```
var x = document.getElementsByTagName("p");
```

- Finding HTML elements by class name

```
var x = document.getElementsByClassName("intro");
```

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

Finding HTML Elements

- Finding HTML elements by CSS selectors

```
var x = document.querySelectorAll("p.intro");
```

- Finding HTML elements by HTML object collections

```
var x = document.forms["frm1"];
var text = "";
var i;
for (i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

Finds the form element id = ‘frm1’

Manipulating HTML Content

- Use **innerHTML** property to modify the content of an HTML element

```
document.getElementById("p1").innerHTML = "Game of Thrones";

var element = document.getElementById("id01");
element.innerHTML = "Game of Thrones";
```

- HTML attributes can be changed using the **attribute**

```
<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>
```

Manipulating CSS Content

- CSS content can be modified using the style property

```
document.getElementById("p2").style.color = "blue";
```

Events

- Javascript DOM functions can be used to react to HTML events

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

displayDate is JS function

- Other events:
  - **onload**, **onunload**
  - **onchange**
  - **onmouseover**, **onmouseout**
  - **onmousedown**, **onmouseup**

Event Listener

- The **`addEventListener()`** method attaches an event handler to the specified element
- Does not override existing event handlers
- Many even handlers can be added
- The event handler can be removed using **`removeEventListener()`**

```
<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```

Any DOM event
https://www.w3schools.com/jsref/dom_obj_event.asp

Event Bubbling and Event Capturing

- Event propagation is a way of defining the element order when an event occurs.
  - **If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?**
- In *bubbling* the inner most element's event is handled first and then the outer
- In *capturing* the outer most element's event is handled first and then the inner

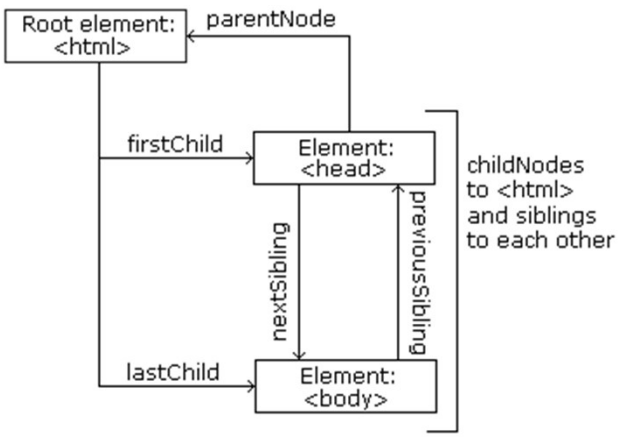
```
addEventListener(event, function, useCapture);
```

Default is False, in the bubbling mode.

Navigation

- The DOM tree can be navigated using the node relationships
- The nodes in the node tree have a hierarchical relationship
  - The terms parent, child, and sibling are used to describe the relationships.
  - In a node tree, the top node is called the root (or root node)
  - Every node has exactly one parent, except the root (which has no parent)
  - A node can have a number of children
  - Siblings (brothers or sisters) are nodes with the same parent

```
<html>

  <head>
    <title id=‘demo’>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



Navigating Between Nodes

- The following node properties can be used to navigate between nodes with JavaScript:
  - **parentNode**
  - **childNodes[nodenumber]**
  - **firstChild**
  - **lastChild**
  - **nextSibling**
  - **previousSibling**

```
var myTitle = document.getElementById("demo").innerHTML;
var myTitle = document.getElementById("demo").firstChild.nodeValue;
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

All these access **DOM Tutorial** in text tag

Navigating Between Nodes

- The full document can be accessed using the following properties
  - **`document.body`** - The body of the document
  - **`document.documentElement`** - The full document
- The node related properties are as follows
  - **`nodeValue`** property specifies the value of a node
  - **`nodeType`** property is read only. It returns the type of a node
    ELEMENT_NODE, TEXT_NODE, DOCUMENT_NODE, COMMENT_NODE …
  - **`nodeName`** property specifies the name of a node
    Same as tag name, attribute name, #text, #document respectively

Creating and Removing Nodes

- To add a new element you need to create the element node and then use **appendChild()** or **insertBefore()** to add the element

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Creating Nodes

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para, child);
</script>
```

Removing a Child Node

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

Replacing Nodes

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

Collections and Node Lists

- An **HTMLCollection** object is an array-like list of HTML elements
- The **length** property defines the number of elements

```
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = myCollection.length;
```

- A **NodeList** object is a list of nodes extracted from a document

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

Returns a node list

Animation – HTML/CSS/JS Integrated Example

```html
<div id ="container">
  <div id ="animate">My animation</div>
</div>
<p><button onclick="myMove()">Click Me</button></p>
```

```css
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background: red;
}
```

```js
function myMove() {
  var elem = document.getElementById("animate");
  var pos = 0;
  var id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + 'px';
      elem.style.left = pos + 'px';
    }
  }
}
```

Browser Object Model - BOM

- BOM allows JS to interact with the browser
- No official standards
- Modern browser have implemented almost same methods and properties for JS interactivity

The Window Object

- It represents the browser's window and supported by all browsers
- Window Size:
  - `window.innerHeight` - the inner height of the browser window (in pixels)
  - `window.innerWidth` - the inner width of the browser window (in pixels)
- Other window operations
  - `window.open()` - open a new window
  - `window.close()` - close the current window
  - `window.moveTo()` - move the current window
  - `window.resizeTo()` - resize the current window

The Window Screen Object

- It contains information of the user screen
- Properties include the following:
  - `screen.width`
  - `screen.height`
  - `screen.availWidth`
  - `screen.availHeight`
  - `screen.colorDepth`
  - `screen.pixelDepth`

Returns the width and height of visitors screen

Window Location

- Can be used to get the current page address
- Also, redirect to a new page
- Examples
  - `window.location.href` returns the href (URL) of the current page
  - `window.location.hostname` returns the domain name of the web host
  - `window.location.pathname` returns the path and filename of the current page
  - `window.location.protocol` returns the web protocol used (http: or https:)
  - `window.location.assign()` loads a new document

Window History

- The `window.history` object contains the browsers history
- To protect the privacy of the users, there are limitations to how JS can access this object.
- Some methods:
  - `history.back()` - same as clicking back in the browser
  - `history.forward()` - same as clicking forward in the browser

Navigator

- The `window.navigator` object contains information about the visitor's browser
- Examples:
  - `navigator.cookieEnabled` – returns if cookies are enabled
  - `navigator.appName` – returns the application name of the browser
  - `navigator.product` – returns the product name of the browser
  - `navigator.appVersion` – returns the browser version
  - `navigator.platform` – returns the browser OS
  - `navigator.language` – returns the browser's language
  - `navigator.onLine` – returns if the browser is online

Pop-ups

- JS has three kinds of information popup boxes

```
alert("I am an alert box!");

confirm("Press a button!")

prompt("Please enter your name", "Harry Potter");
```

Timing

- JS can be executed in time intervals, this is called timing events
- The two key methods to use with JavaScript are:
  - **`setTimeout(function, milliseconds)`**
    Executes a function, after waiting a specified number of milliseconds
    Cleared by **`clearTimeout()`**
  - **`setInterval(function, milliseconds)`**
    Same as setTimeout(), but repeats the execution of the function continuously
    Cleared by **`clearInterval()`**

Timing

```
<p id="demo"></p>

<button onclick="clearInterval(myVar)">Stop time</button>

<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
  var d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
```

Study clock created by timing event

Cookies

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his/her name.
- Cookies are saved in name-value pairs like:

```
username = John Doe
```

Cookies

- Creating a cookie

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

- Reading a cookie

```
var x = document.cookie;
```

Will return all cookies in one string, write a function to search for a specific value

- Change a cookie

```
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

- Deleting a cookie

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

Set to some old date

Javascript Quiz

- https://www.w3schools.com/js/js_quiz.asp

jQuery

https://api.jquery.com/
https://www.tutorialsteacher.com/jquery/jquery-tutorials
https://www.tutorialrepublic.com/jquery-examples.php

https://www.educba.com/javascript-vs-jquery/

Please, stop torturing yourself with **document.getElementByWhatever** and start using **$('[CSS_SELECTOR]').doSomething()**

jQuery

- jQuery is a JavaScript Library
- It's light weight and works with "**write less, do more**" philosophy
- It contains following features:
  - **Unobtrusive** HTML/DOM, CSS manipulation
  - HTML event methods
  - Effects and animations
  - AJAX
  - Utilities

jQuery

- Basic Syntax: **$(*selector*).*action*()**
  - **Example:** `$(this).hide()` – hides the current element
  - `$("p").hide()` – hides all <p> element

- Including jQuery in the project
  - Download and refer to it

    ```
    <script src="jquery-3.5.1.min.js"></script>
    ```

  - Use the CDN

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

JS Vs jQuery

JavaScript

```
var myElement = document.getElementById("id01");

var myElements = document.getElementsByTagName("p");

var myElements = document.getElementsByClassName("intro");

var myElements = document.querySelectorAll("p.intro");
```

jQuery

```
var myElement = $("#id01");

var myElements = $("p");

var myElements = $(".intro");

var myElements = $("p.intro");
```

jQuery for HTML Manipulation

Setting Values

```
$("#test1").text("Hello world!");
$("#test2").html("<b>Hello world!</b>");
$("#test3").val("Hello World");
$("#w").attr("href", "/jquery")
```

Getting Values

```
var x = $("#test1").text();
var x = $("#test2").html();
var x = $("#test3").val();
var x = $("#w").attr("href")
```

```
<p><a href="" title="" id="w">jQuery</a></p>
```

```
$("button").click(function(){
  $("#w").attr({
    "href" : "/jquery/",
    "title" : "jQuery Tutorial"
  });
});
```

The jQuery ready()

- The .ready() method offers a way to run JavaScript code as soon as the page's Document Object Model (DOM) becomes safe to manipulate.
- This will often be a good time to perform tasks that are needed before the user views or interacts with the page, for example to add event handlers and initialize plugins.

```
$( document ).ready(function() {
    // Handler for .ready() called.
  });
```

Adding Elements

- Adding of elements can be done using
  - **append()** and **prepend()**
  - **before()** and **after()**

```
function appendText() {
  var txt1 = "<p>Text.</p>";                // Create element with HTML
  var txt2 = $("<p></p>").text("Text.");    // Create with jQuery
  var txt3 = document.createElement("p");   // Create with DOM
  txt3.innerHTML = "Text.";
  $("body").append(txt1, txt2, txt3);       // Append the new elements
}
```

Removing Elements

- The empty() method removes the child elements

```
$("#div1").empty();
```

- The remove() filter the elements to be removed

```
$("p").remove(".test, .demo");
```

jQuery for CSS Manipulation

- jQuery has several methods for CSS manipulation. We will look at the following methods:
  - `addClass()` - Adds one or more classes to the selected elements
  - `removeClass()` - Removes one or more classes from the selected elements
  - `toggleClass()` - Toggles between adding/removing
  - `css()` - Sets or returns the style attribute

```html
<div id="div1">This is some text.</div>
<div id="div2">This is some text.</div>
```

```style
<style>
.important {
  font-weight: bold;
  font-size: xx-large;
}

.blue {
  color: blue;
}
</style>
```

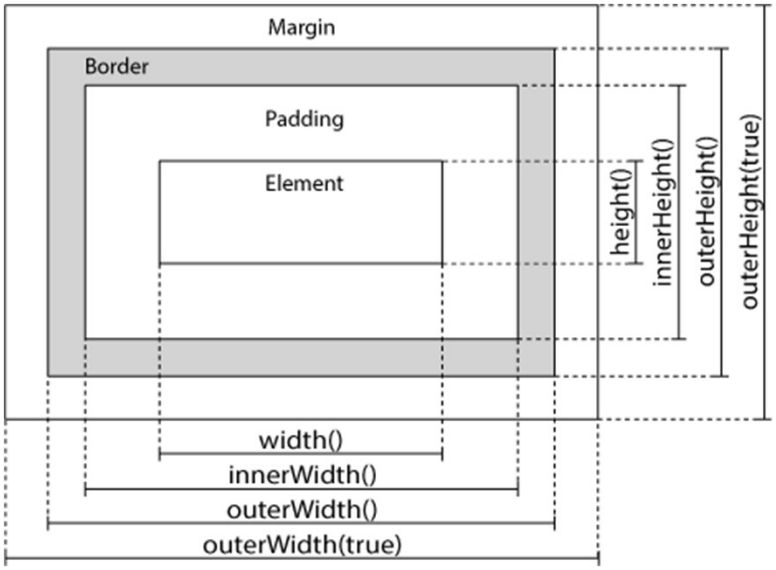
```script
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").addClass("important blue");
  });
});
</script>
```

Using css()

```
$("p").css("background-color", "yellow");
$("p").css({"background-color": "yellow", "font-size": "200%"});
```

jQuery Dimensions

- jQuery has several important methods for working with dimensions:
  - `width()`
  - `height()`
  - `innerWidth()`
  - `innerHeight()`
  - `outerWidth()`
  - `outerHeight()`



jQuery Events

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

The $(document).ready() method allows us to execute a function when the document is fully loaded.

```
<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
</script>
```

jQuery Effects

- Hide and Show

```
$("p").hide();
$("p").show();
$("p").toggle();
```

Hide/show/toggle(*speed,callback*); speed → "slow", "fast", milliseconds, the callback function executes once the process completes

- Fade

```
$("#div1").fadeIn();
$("#div2").fadeIn("slow");
$("#div3").fadeIn(3000);
```

```
$("#div1").fadeOut();
$("#div2").fadeOut("slow");
$("#div3").fadeOut(3000);
```

```
$("#div1").fadeToggle();
$("#div2").fadeToggle("slow");
$("#div3").fadeToggle(3000);
```

```
$("#div1").fadeTo("slow", 0.15);
$("#div2").fadeTo("slow", 0.4);
$("#div3").fadeTo("slow", 0.7);
```

Callbacks can be written for all functions
fadeTo(*speed,opacity,callback*);

jQuery Effects

- Slide

```
$("#panel").slideDown();
$("#panel").slideUp();
$("#panel").slideToggle();
```

- Animate

```
$("button").click(function(){
  $("div").animate({
    left: '250px',
    opacity: '0.5',
    height: '150px',
    width: '150px'
  });
});
```

```
$("button").click(function(){
  var div = $("div");
  div.animate({height: '300px', opacity: '0.4'}, "slow");
  div.animate({width: '300px', opacity: '0.8'}, "slow");
  div.animate({height: '100px', opacity: '0.4'}, "slow");
  div.animate({width: '100px', opacity: '0.8'}, "slow");
});
```

Relative values can be used: height: “+=150” or pre-defined values can be specified: “show”, “hide”, “toggle”. Optionally speed and callback can be specified

jQuery Effects

- **stop() –** To stop an animation or effect before it is finished

```
$(selector).stop(stopAll,goToEnd);
```

stopAll → whether the animation queue should be cleared or not, default false
goToEnd → whether to complete current animation or not, default false

- Chaining – chain together actions and methods

```
$("#p1").css("color", "red")
  .slideUp(2000)
  .slideDown(2000);
```

jQuery Traversing

- Traversing allow you to move through the HTML elements

```
$("span").parent();
$("span").parents();
$("span").parents("ul");
$("span").parentsUntil("div");
```

```
$("div").children();
$("div").children("p.first");
$("div").find("*");
$("h2").siblings("p");
$("h2").next();
$("h2").nextAll();
$("h2").nextUntil("h6");
```

The prev(), prevAll() and prevUntil() work with reverse functionality

```
$("div").first();
$("div").last();
$("p").eq(1);
$("p").filter(".intro");
$("p").not(".intro");
```

```
<script>
$(document).ready(function(){
  $("h2").next().css({"color": "red", "border": "2px solid red"});
});
</script>
```