-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

**JS Best Practices**

|  |  |  |
| --- | --- | --- |
| Global Variables | Global variables and functions can be overwritten by other scripts => Avoid |  |
| Local Variables | Always declare with the **var** keyword or the **let** keyword |  |
| Declaration | Should be at the top of each script or function. | // Declare at the beginning let firstName, lastName, price, discount, fullPrice;  // Use later firstName = "John"; lastName = "Doe"; |
| Initialize Variables | Should be initialized when you declaring | // Declare and initiate at the beginning let firstName = "", let lastName = "", |
| Declare Objects | Declaring with **const** to prevent any accidential change of type | const car = {type:"Fiat", model:"500", color:"white"}; car = "Fiat";      // Not possible |
| Declare Arrays | Declaring with **const** to prevent any accidential change of type | const cars = ["Saab", "Volvo", "BMW"]; cars = 3;    // Not possible |
| new Object() | Don't Use new Object()  - Use "" instead of new String()  - Use 0 instead of new Number()  - Use false instead of new Boolean()  - Use {} instead of new Object()  - Use [] instead of new Array()  - Use /()/ instead of new RegExp()  - Use function (){} instead of new Function() | let x1 = "";             // new primitive string let x2 = 0;              // new primitive number let x3 = false;          // new primitive boolean const x4 = {};           // new object const x5 = [];           // new array object const x6 = /()/;         // new regexp object const x7 = function(){}; // new function object |
| Automatic Type Conversions | Beware of Automatic Type Conversions:  A variable can change its data type | let x = "Hello";     // typeof x is a string x = 5;               // changes typeof x to a number  let x = 5 + "7";     // typeof x is a string  "Hello" - "Dolly"    // returns NaN |
| Comparison | Using === operator forces comparison of values and type | 0 == "";        // true 0 === "";       // false |
| Parameter Defaults | Missing argument, the value is set to undefined.  => It's a good practice to assign default values to arguments. | function myFunction(x, y) {   if (y === undefined) {     y = 0;   } } |
| Switches with Defaults | Always end switch statements with a default.  Even if you think there is no need for it. | switch (new Date().getDay()) {   case 0:     day = "Sunday";     break;  // some code    case 6:     day = "Saturday";     break;   default:     day = "Unknown"; } |
| Primative Values  or Objects | Not treat numbers, strings, or booleans as objects.  slow down execution speed, and produces nasty side effects: | let x = "John";              let y = new String("John"); (x === y) // is false because x is a string and y is an object. |
|  |  | let x = new String("John");              let y = new String("John"); (x == y) // is false because you cannot compare objects. |
| eval() | it should not be necessary to use it. Because it allows arbitrary code to be run, it also represents a security problem. |  |

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

**JS Common Mistakes**

|  |  |  |
| --- | --- | --- |
| Assignment Operator | assignment operator (=), instead of a comparison operator (==) in an if statement. | let x = 0; if (x = 10) // return true  let x = 0; if (x = 0) // return false because 0 is false |
| Loose or Strict Comparison | switch statements use strict comparison: | let x = 10; switch(x) {   case "10": alert("Hello"); } // not work |
| Addition & Concatenation | **Addition** is about adding **numbers**.  **Concatenation** is about adding **strings**. | let y = 10; y += "5";   // Now y is "105" |
| Floats | **Problem**: all programming languages have difficulties with precise floating point values: | let x = 0.1; let y = 0.2; let z = x + y  // the result in z will not be 0.3 |
|  | **Solution**: it helps to multiply and divide: | let z = (x \* 10 + y \* 10) / 10;  // z will be 0.3 |
| Breaking a JS String | Breaking a statement in the middle of a string will not work: | let x = "Hello World!"; // not work  // using \ instead  let x = "Hello \ World!"; |
| Semicolon | Misplacing Semicolon can make the code runing without checking | let x = 5;  if (x == 19);  {  document.getElementById("demo").innerHTML = "Hello";  } |
| Return Statement | By default, JS close a statement automatically at the end of a line =>  semicolon is optional in JavaScript. | function myFunction(a) {   let // not a complete statement => JS check the next line;   power = 10;   return a \* power; } // return: 550 |
|  | **Solution**: Never break a return statement. | function myFunction(a) {   let // not a complete statement => JS check the next line;   power = 10;    return // a complete statement => the same as return;    a \* power; } // return: undefined |
| Accessing Arrays  with Named Indexes | In many languages, arrays use named indexes:  In JS, arrays use numbered indexes: | const person = []; person[0] = "John"; person[1] = "Doe"; person[2] = 46; person.length;       // person.length will return 3 person[0];           // person[0] will return "John" |
|  | In JS, objects use named indexes. | const person = []; person["firstName"] = "John"; person["lastName"] = "Doe"; person["age"] = 46; person.length;      // person.length will return 0 person[0];          // person[0] will return undefined |
| Ending Definitions  with a Comma | Trailing commas in object and array is legal  But Internet Explorer 8 will crash => not neccessary | person = {"firstName":"John", "lastName":"Doe", "age":46,}  points = [40, 100, 1, 5, 25, 10,]; |
| Undefined  is Not Null | JS objects, variables, properties, methods can be undefined.  Empty JS objects can have the value null.  => difficult to test if an object is empty. |  |
|  | **Best solution**: test for not undefined before you can test for not null: | if (typeof myObj !== "undefined" && myObj !== null) |

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

**JS Performance**

|  |  |  |
| --- | --- | --- |
| General | How to speed up your JavaScript code. |  |
| Reduce Activity  in Loops | Statements or assignments should be placed outside the loop to make the loop run faster. | // bad  for (let i = 0; i < arr.length; i++) {  // better  let l = arr.length; for (let i = 0; i < l; i++) { |
| Reduce DOM Access | **Problem**: accessing the HTML DOM is very slow, compared to other JS statements. |  |
|  | **Solution**: if you expect to access a DOM element several times, access it once, and use it as a local variable: | const obj = document.getElementById("demo");  obj.innerHTML = " Hello"; |
| Reduce DOM Size | Keep the number of elements in the HTML DOM small. |  |
| Avoid Unnecessary Variables | Don't create new variables if you don't plan to save values. | // bad  let fullName = firstName + " " + lastName; document.getElementById("demo").innerHTML = fullName;  // better  document.getElementById("demo").innerHTML = firstName + " " + lastName; |
| Delay JS Loading | 1. Putting your scripts at the bottom of the page body lets the browser load the page first. |  |
|  | 2. Use **defer="true"** in the script tag.  (it specifies that the script should be executed after the page has finished parsing, but it only works for external scripts.) |  |
|  | 3. add your script to the page by code, after the page has loaded: | <script> window.onload = function() {   const element = document.createElement("script");   element.src = "myScript.js";   document.body.appendChild(element); }; </script> |
| Avoid Using **with** | **with** keyword is not allowed in strict mode.  (It has a negative effect on speed. It also clutters up JS scopes) |  |

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**JS Web APIs**

**Web API Intro**

|  |  |  |
| --- | --- | --- |
| API | **A**pplication **P**rogramming **I**nterface for the Web  - can extend the functionality of a web browser.  - can extend the functionality of a web server. |  |
| Browser APIs | built-in Web APIs to support complex operations, and to help accessing data. | // Get the latitude and longitude of the user's position:  const myElement = document.getElementById("demo");  function getLocation() {   if (navigator.geolocation) {     navigator.geolocation.getCurrentPosition(showPosition);   } else {     myElement.innerHTML = "Geolocation is not supported by this browser.";   } }  function showPosition(position) {   myElement.innerHTML = "Latitude: " + position.coords.latitude +   "<br>Longitude: " + position.coords.longitude; } |
| Third Party APIs | not built into your browser, must download from the web | YouTube API - to display videos on a web site.  Twitter API - display Tweets on a web site.  Facebook API - display Facebook info on a web site. |

**Web Validation API**

|  |  |  |
| --- | --- | --- |
| Constraint Validation DOM Methods | | |
| checkValidity() | Returns true if an input element contains valid data. | // basic code  <input id="id1" type="number" min="100" max="300" required>  <button onclick="myFunction()">OK</button>  <p id="demo"></p>  // check input  <script>  function myFunction() {  const **inpObj** = document.getElementById("id1");  if (!inpObj.**checkValidity**()) {  document.getElementById("demo").innerHTML = inpObj.validationMessage;  } else {  document.getElementById("demo").innerHTML = "Input OK";  }  } |
| setCustomValidity() | Sets the validationMessage property of an input element. |  |
| Constraint Validation DOM Properties | | |
| validity | boolean properties related to the validity of an input element. |  |
| validationMessage | the message a browser will display when the validity is false. |  |
| willValidate | Indicates if an input element will be validated. |  |
| Validity Properties: contains a number of properties related to the validity of data: | | |
| customError | Set to true, if a custom validity message is set. | // greater than 100, display a message:  <input id="id1" type="number" max="100"> <button onclick="myFunction()">OK</button>  <p id="demo"></p>  <script> function myFunction() {   let text = "Value OK";   if (document.getElementById("id1").**validity**.**rangeOverflow**) {     text = "Value too large";   } } </script>  // input 1000 => result: Value too large. |
| 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. |

**Web History API**

|  |  |  |
| --- | --- | --- |
| General | provide easy methods to access the **windows**.**history** object. |  |
| forward() | Loads the next URL in the history list | <button onclick="myFunction()">Go Back</button>  <script> function myFunction() {   window.history.**forward**(); } </script> |
| back() | loads the previous URL in the windows.history list.  (the same as clicking the "back arrow") | <button onclick="myFunction()">Go Back</button>  <script> function myFunction() {   window.history.**back**(); } </script> |
| go() | loads a specific URL from the history list: | <button onclick="myFunction()">Go Back 2 Pages</button>  <script> function myFunction() {   window.history.**go**(-2); // back two pages } </script> |
| length | Returns the number of URLs in the history list | var x = history.length;  document.getElementById("demo").innerHTML = x; |

**Web Storage API**

|  |  |  |
| --- | --- | --- |
| General | a simple syntax for storing and retrieving data in the browser. | localStorage.setItem("name", "John Doe"); localStorage.getItem("name"); |
| localStorage | provides access to a local storage for a particular Web Site  (the data is stored with no expiration date, and will not be deleted when the browser is closed.) |  |
| sessionStorage | identical to the localStorage object, but stxores data for one session. (the data is deleted when the browser is closed) |  |
| Methods | | |
| setItem() | stores a data item in a storage.  localStorage.**setItem**(*name*,*value*);  sessionStorage.**setItem**(*name*,*value*); | localStorage.setItem("name", "John Doe");  sessionStorage.setItem("name", "John Doe"); |
| getItem() | retrieves a data item from the storage.  localStorage.**getItem**(*name*);  sessionStorage.**getItem**(*name*); | localStorage.getItem("name");  sessionStorage.getItem("name"); |
| key() | returns name of the key with the specified index.  localStorage.**key**(*index*)  sessionStorage.**key**(*index*) | var x = localStorage.key(0); |
| removeItem() | removes the specified Storage Object item.  localStorage.**removeItem**(*keyname*)  sessionStorage.**removeItem**(*keyname*) | localStorage.removeItem("mytime"); |
| clear() | removes all the Storage Object item for this domain.  localStorage.**clear**()  sessionStorage.**clear**() | sessionStorage.clear(); |
| Properties | | |
| length | returns the number of items stored in the Storage Object  localStorage.**length**;  sessionStorage.**length**; | var x = localStorage.length; |

**Web Woker API**

|  |  |  |
| --- | --- | --- |
| Web Worker | a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. |  |
| Checking | check whether the user's browser supports it: | if (typeof(Worker) !== "undefined") {   // Yes! Web worker support!   // *Some code.....* } else {   // Sorry! No Web Worker support.. } |
| Creating a Web Worker File |  | // create a script that counts.  let i = 0;  function timedCount() {   i ++;   **postMessage**(i); // the most important part   setTimeout("timedCount()",500); }  timedCount(); |
| Create a Web Worker Object | checks if the worker already exists,  if not - it creates a new web worker object  and runs the code in "demo\_workers.js": | if (typeof(w) == "undefined") {   w = new Worker("demo\_workers.js"); } |
|  | add an "onmessage" event listener to the web worker. | w.onmessage = function(event){   document.getElementById("result").innerHTML = event.data; }; |
| Terminate a Web Worker | use the **terminate**() method: | w.terminate(); |
| Reuse the Web Worker | If you set the worker variable to undefined, after it has been terminated, you can reuse the code: | w = undefined; |
| Full Code |  | <!DOCTYPE html> <html> <body>  <p>Count numbers: <output id="result"></output></p> <button onclick="startWorker()">Start Worker</button> <button onclick="stopWorker()">Stop Worker</button>  <script> let w;  function startWorker() {   if (typeof(w) == "undefined") {     w = new Worker("demo\_workers.js");   }   w.onmessage = function(event) {     document.getElementById("result").innerHTML = event.data;   }; }  function stopWorker() {   w.terminate();   w = undefined; } </script>  </body> </html> |

**Web Fetch API**

|  |  |  |
| --- | --- | --- |
| General | allows web browser to make HTTP requests to web servers. | fetch(file) .then(x => x.text()) .then(y => myDisplay(y)); |
|  | Since Fetch is based on async and await, the example above might be easier to understand like this: | async function getText(file) {   let x = await fetch(file);   let y = await x.text();   myDisplay(y); } |
|  | Or even bettter: Use understandable names instead of x and y: | async function getText(file) {   let myObject = await fetch(file);   let myText = await myObject.text();   myDisplay(myText); } |

**Web Geolocation API**

|  |  |  |
| --- | --- | --- |
| Geolocation API | used to get the geographical position of a user. |  |
| getCurrentPosition() | used to return the user's position. | // returns the latitude and longitude of the user's position:  const x = document.getElementById("demo"); function getLocation() { // Check if Geolocation is supported  // supported, run the getCurrentPosition() method   if (navigator.geolocation) {     navigator.geolocation.getCurrentPosition(showPosition);  // not supported, display a message to the user    } else {     x.innerHTML = "Geolocation is not supported by this browser.";   } } // outputs the Latitude and Longitude function showPosition(position) {   x.innerHTML = "Latitude: " + position.coords.latitude +   "<br>Longitude: " + position.coords.longitude; } |
| Errors and Rejections | the second parameter of the **getCurrentPosition**() method is used to handle errors. It specifies a function to run if it fails to get the user's location: | function showError(error) {   switch(error.code) {     case error.PERMISSION\_DENIED:       x.innerHTML = "User denied the request for Geolocation."       break;     case error.POSITION\_UNAVAILABLE:       x.innerHTML = "Location information is unavailable."       break;     case error.TIMEOUT:       x.innerHTML = "The request to get user location timed out."       break;     case error.UNKNOWN\_ERROR:       x.innerHTML = "An unknown error occurred."       break;   } } |
| Displaying the Result in a Map |  | function showPosition(position) {   let latlon = position.coords.latitude + "," + position.coords.longitude;    let img\_url = "https://maps.googleapis.com/maps/api/staticmap?center=   "+latlon+"&zoom=14&size=400x300&sensor=false&key=YOUR\_KEY";    document.getElementById("mapholder").innerHTML = "<img src='"+img\_url+"'>"; } |
| watchPosition() | Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car). | <script> const x = document.getElementById("demo"); function getLocation() {   if (navigator.geolocation) {     navigator.geolocation.watchPosition(showPosition);   } else {     x.innerHTML = "Geolocation is not supported by this browser.";   } } function showPosition(position) {   x.innerHTML = "Latitude: " + position.coords.latitude +   "<br>Longitude: " + position.coords.longitude; } </script> |
| clearWatch() | Stops the watchPosition() method. |  |

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**JS AJAX**

**AJAX Intro**

|  |  |  |
| --- | --- | --- |
| General | **A**synchronous **J**avaScript **A**nd **X**ML: a combination of  - a browser built-in **XMLHttpRequest** object (to request data from a web server)  - JS and HTML DOM (to display or use the data) | AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text. |
| How AJAX Works | 1. An event occurs in a web page (the page is loaded, a button is clicked)  2. An XMLHttpRequest object is created by JS  3. The XMLHttpRequest object sends a request to a web server  4. The server processes the request  5. The server sends a response back to the web page  6. The response is read by JS  7. Proper action (like page update) is performed by JS |  |
| Fetch API | Modern Browsers can use Fetch API instead of the XMLHttpRequest Object.  - The Fetch API interface allows web browser to make HTTP requests to web servers.  - If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way. |  |

**AJAX XMLHttpRequest Object**

|  |  |  |
| --- | --- | --- |
| General | used to exchange data with a web server behind the scenes. | => possible to update parts of a web page, without reloading the whole page. |
| Creating | *variable*= new XMLHttpRequest();W  (All modern browsers have a built-in XMLHttpRequest object) |  |
| Define a Callback Function | A callback function is a function passed as a parameter to another function.   the callback function should contain the code to execute when the response is ready. | // Define a callback function xhttp.onload = function() {   // Here you can use the Data } |
| Send a Request to the Server | use the open() and send() methods of the XMLHttpRequest object | // Send a request xhttp.open("GET", "ajax\_info.txt"); xhttp.send(); |
| Access Across Domains | - For security reasons, modern browsers do not allow access across domains. | => both the web page and the XML file it tries to load, must be located on the same server. |
| XMLHttpRequest Object Methods | | |
| new XMLHttpRequest() | create a new XMLHttpRequest object |  |
| abort() | cancel the current request |  |
| getAllResponseHeaders() | return header information |  |
| getResponseHeader() | Returns specific header information |  |
| open(method, url, async, user, psw) | specify the request  url: the file location  async: true (asynchronous) or false (synchronous)  user: optional user name  psw: optional password |  |
| send() | Sends the request to the server  Used for GET requests |  |
| send(string) | Sends the request to the server.  Used for POST requests |  |
| setRequestHeader() | add a label/value pair to the header to be sent |  |
| XMLHttpRequest Object Properties | | |
| onload | Defines a function to be called when the request is recived (loaded) |  |
| onreadystatechange | Defines a function to be called when the readyState property changes |  |
| readyState | Holds the status of the XMLHttpRequest.  0: request not initialized  1: server connection established  2: request received  3: processing request  4: request finished and response is ready |  |
| responseText | Returns the response data as a string |  |
| responseXML | Returns the response data as XML data |  |
| status | Returns the status-number of a request  200: "OK"  403: "Forbidden"  404: "Not Found"  https://www.w3schools.com/tags/ref\_httpmessages.asp |  |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |  |

**AJAX Request**

|  |  |  |
| --- | --- | --- |
| General | XMLHttpRequest object is used to request data from a server. |  |
| Send a Request | **open**(*method, url, async*) - specify the type of request  *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous) | xhttp.open("GET", "ajax\_info.txt", true); |
|  | **send**() - send the request to the server (used for GET) | xhttp.send(); |
|  | **send**(*string*) - send the request to the server (used for POST) | xhttp.send("fname=Henry&lname=Ford"); |

**AJAX Response**

|  |  |  |
| --- | --- | --- |
| Sever Response Properties | | |
| responseText | returns the server response as a JS string, and you can use it accordingly: | document.getElementById("demo").innerHTML = xhttp.**responseText**; |
| responseXML | returns the server response as an XML DOM object.  (Using this property you can parse the response as an XML DOM object:) | // Request the file [cd\_catalog.xml](https://www.w3schools.com/js/cd_catalog.xml) and parse the response:  const xmlDoc = xhttp.responseXML; const x = xmlDoc.getElementsByTagName("ARTIST");  let txt = ""; for (let i = 0; i < x.length; i++) {   txt += x[i].childNodes[0].nodeValue + "<br>"; } document.getElementById("demo").innerHTML = txt;  xhttp.open("GET", "cd\_catalog.xml"); xhttp.send(); |
| Server Response Methods | | |
| getResponseHeader() | return specific header information from the server resource | const xhttp = new XMLHttpRequest(); xhttp.onload = function() {     document.getElementById("demo").innerHTML =     this.getResponseHeader("Last-Modified"); } xhttp.open("GET", "ajax\_info.txt"); xhttp.send(); |
| getAllResponseHeaders() | return all the header information from the server resource | const xhttp = new XMLHttpRequest(); xhttp.onload = function() {     document.getElementById("demo").innerHTML =     this.getAllResponseHeaders(); } xhttp.open("GET", "ajax\_info.txt"); xhttp.send(); |

**AJAX XML File**

|  |  |
| --- | --- |
| General |  |

**AJAX Example**

|  |  |  |
| --- | --- | --- |
| XML file | AJAX can be used for interactive communication with an XML file. |  |
| PHP | AJAX is used to create more interactive applications. |  |
| ASP | AJAX is used to create more interactive applications. |  |
| ASP | AJAX can be used for interactive communication with a database. |  |