**Javascript einbinden in HTML-head:** <script language="javascript" type="text/javascript" src="script.js"></script>

**Output mit:**

* innerHTML schreibt direkt in den HTML-Text, hier in <p>

<p id="demo">Display the result here.</p> ...

document.getElementById("demo").innerHTML = "Hello World";

**Hoisting**: Ist das default-Verhalten von JS, bei dem Deklarationen an die Spitze des aktuellen Scope „gehoben“ werden.

* gültig für Funktionen- und Variablen-Deklarationen

**Design Pattern:** Ein formaler Weg zur Dokumentation eines Design Problems.

**Pattern Language:** Eine organisierte Sammlung von Design Patterns zu einem bestimmten Problemgebiet.

# Values

**var** → keine Info über Datentyp: wird zur Laufzeit evaluiert. **Strings** mit „“ od. ‚’ und **Nummern** mit oder ohne Dezimalzeichen „.“ **Primitive Typen** sind strings, numbers, booleans, null, undefinded, symbol → alle anderen Values sind Objekte.

Länge: mit var length = variablenName.length;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| \' | single quote | \n | new line | \b | backspace |
| \" | double quote | \r | carriage return | \f | form feed |
| \\ | backslash | \t | tab |  |  |

**Variablendeklaration** mit var deklariert → Deklaration ohne var sind automatisch global

* Zuweisen mit = += -= \*= etc. (assignment operators)
* Arithmetische Operatoren: + - \* / % ++ --
* Kommentar mit // oder /\* \*/
* Statments: Abschliessen mit ; , Gruppieren mit {}
* Wert vergleichen mit == (equal) → macht evt. Cast
* Wert und Typ vergleichen mit === (strict equal)

**Array**: var cars = [„Volvo“, „Saab“, „Fiat“];

# Object

* Fast alles in JS ist ein Objekt (immer: Funktionen, Arrays, Objekte, alles was mit Object Constructor aufgerfufen wird. Manchmal: Booleans, primitive Datentypen).
* nur Referenzen auf Objekte werden weitergereicht, nie Kopien
* null ist in JS ein Object (null == undefined liefert true)

Objekt erstellen mit **Object** **Literal**

var objectName = { firstName: “Jon“, age: 50, //name/value-Paar fullName : function() { // Objektmehtode

return this.firstName + " " + this.lastName; }};

Objekt erstellen mit Schlüsselwort **new**

var person = new Object();

person.firstname = „John“; …

**Constructor**

var Person = function (first, last, age) {

this.firstname = first; … }

var john = new Person („John“, „Deer“, 50);

Aufrufen von Properties auf 2 Arten, wobei || (optional) benutzt wird um default-Value zurück zu geben falls Value undefined ist):

1. objectName.propertyName || default; // bevorzugt
2. objectName["propertyName"] || default; → Objektmethoden analog aufrufen:
3. objectName.objectMehtod()

**Prototype** //gibt allen Person Instanzen eine getName-Funktion

Person.prototype.getName = function() {

return this.first + „ „ + this.last };

var fruit = { name: „name“ } //normales Objekt

var Berry = function(name) { this.name = name}; //Subklasse

Berry.prototype = fruit; //Berry erbt von fruit

var strawberry = new Fruit(„strawberry“);

**this**

* this aufgerufen in einer Funktion ist das Objekt, welches die Funktion besitzt
* this aufgerufen in einem Objekt ist das Objekt selber
* this in einem Object Constructor hat kein value, sondern verlinkt auf das neue Objekt.

# Function

**Klassisch:** function nameOfFunction(arg1, arg2) { this.arg1 = arg1; this.arg2 = arg2; }

**Arrow Notation:** functionTwo = () => { code }

**Anonyme Funktion**: function(parameter) { code } (); ODER (parameter) => { code }

→ Optional: Klammern, wenn nur 1 Parameter vorhanden ist. Return, wenn F. auf einer Zeile.

**Lokale Funktion:** Anonyme F. einer Variable zuweisen → execute at run time. var f = function() {…};

* Self invoking: Funktion führt sich selber aus, wenn function von () gefolgt ist.
* as a function: F. gehört nicht zu einem Objekt, resp. F. wird globalem Objekt zugewiesen. Bsp. myFunction();
* as method: Aufruf einer Objektmethode. Bsp. myObject.fullName();
* with a function constructor: erzeugen eines Objekts:

Fehlende Parameter: Werden per default als null übergeben.

**Callback:** Code, welcher zurückgerufen wird. Timeout**:** → sleep(5000); würde Browser für 5s blockieren:

asynchronous approach, using a **callback function**

function hello() { alert('Hi there!'); }

setTimeout(hello, 5000);

asynchronous approach, using an **anonymous function**

setTimeout(function() { alert('Hi there!') }, 5000);

# Test Driven Development TDD

**Jasmine Framework**: github.com/jasmine/jasmine → nach „StandAlone“ suchen. Beispiel:

describe("Player", function() { var player, song;

beforeEach(function() { player = new Player();

song = new Song(); });

it("should be able to play a Song", function() {

player.play(song);

expect(player.currentlyPlayingSong).toEqual(song); });

**expect(x).**not.toEqual(x), toContain(y), toBeTruthy(), toBeFalsy() → evaluiert zu false: false, null, 0, „“, undefined, NaN, toBeNull(), Strings: expect("a").toBeLessThan("z"); expect(3.2).toBeCloseTo(3,0);//0=Anz.Dezimalst., toBeGreaterThan(1), toBeLessThan(8),

**Mock Objects** simulieren das Verhalten von Objekten. Verwenden, wenn:

* Isolation des Tests durchbrochen wird
* Resultate nicht-deterministisch sind (zB akt. Zeit)
* Status von Objekt schwierig zu Erstellen/Reproduzieren ist (zB network error)
* Objekt langsam ist (zB Database muss zuerst erstellt werden)
* Objekt existiert noch nicht oder ändert sein Verhalten

**spyOn** ersetzt Funktion durch Spy: spyOn(klassenname, „funktionenname“)

**Create environment:** var env = jasmine.getEnv();

**Run environment:** var currentWindowOnload = window.onload;

window.onload = function() {

if (currentWindowOnload) {

currentWindowOnload();}

htmlReporter.initialize();

env.execute(); };

# jQuery

…ist eine JS Library und enthält: HTML DOM manipulation, CSS manipulation, HTML event methods, Effects and animations, AJAX, Utilities…

Basic **syntax** $(selector).action(): $ sign to define/access jQuery, (selector) to "query (or find)" HTML elements, a jQuery action() to be performed on the element(s)

Verwendet **CSS3-Selektoren**:

* $(this).hide() - hides the current element
* Elementselektor: $("p").hide() - hides all <p> elements
* Classselektor: $(".test") elements with class="test"
* IDSelektor: $("#test")
* Attributselektor: $(„a[type=text]“)

**Child Attribute Selectors:**

|  |  |
| --- | --- |
| \* | Matches any element. |
| E | Matches all element with tag name E. |
| E F | Matches all elements with tag name F that are descendents of E. |
| E>F | Matches all elements with tag name F that are direct children of E. |
| E+F | Matches all elements F immediately preceded by sibling E. |
| E~F | Matches all elements F preceded by any sibling E. |
| E:has(F) | Matches all elements with tag name E that have at least one descendent with tag name F. |
| E.C | Matches all elements E with class name C. Omitting E is the same as \*.C. |
| E#I | Matches element E with id of I. Omitting E is the same as \*#I. |
| E[A] | Matches all elements E with attribute A of any value. |
| E[A=V] | Matches all elements E with attribute A whose value is exactly V. |
| E[A^=V] | Matches all elements E with attribute A whose value begins with V. |
| E[A$=V] | Matches all elements E with attribute A whose value ends with V. |
| E[A\*=V] | Matches all elements E with attribute A whose value contains V. |

**Selecting by Position:**

|  |  |
| --- | --- |
| :first | The first match of the page. li a:first returns the first link also under a list item. |
| :last | The last match of the page. li a:last returns the last link also under a list item. |
| :first-child | The first child element. li:first-child returns the first item of each list. |
| :last-child | The last child element. li:last-child returns the last item of each list. |
| :only-child | Returns all elements that have no siblings. |
| :nth-child(n) | The nth child element. li:nth-child(2) returns the second list item of each list. |
| :nth-child( even / odd) | Even or odd children. li:nth-child(even) returns the even children of each list. |
| :nth-child(Xn+Y) | The nth child element computed by the supplied formula. If Y is 0, it may be omitted. li:nth-child(3n) returns every third item, whereas li:nth-child(5n+1) returns the item after every fifth element. |
| :even and :odd | Even and odd matching elements page-wide. li:even returns every even list item. |
| :eq(n) | The nth matching element. |
| :gt(n) | Matching elements after (and excluding) the nth matching element. |
| :lt(n) | Matching elements before (and excluding) the nth matching element. |

**Custom jQuery Selectors:** :button (alle Buttons), :enabled, :disabled, :checkbox, :checked, …

# HTML-Markup to DOM (Document Object Model) Elements

→ im DOM sind alle HTML Elemente als Objekte definiert

.attr(„attributname“): gibt das erste passende Attribut-Value

* removeAttr()
* attr(„Attribut“, „neuerAttributWert“) //Attributwert ändern
* attr(value: „“, title: „Value ist leer“) //mehrere Attribute

.data()

.swap(): ermöglichst Wechsel zwischen zwei Zuständen

function swap() {

$(„tr).toggleClass(„gefüllt“); }

<table mouseover=“swap();“ onmouseout=“swap();“>

**Holen und Ändern von CSS:**

|  |  |
| --- | --- |
| .css() | Bspw. with von allen Elementen um 20 erhöhen. $("div.expandable").css("width", () => $(this).width + 20 + "px"; ); |
| .html() | Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element. |
| .text() | Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements. |

**Bewegen und Kopieren von Elementen:**

|  |  |
| --- | --- |
| .append() | Insert content, specified by the parameter, to the end of each element in the set of matched elements. |
| .appendTo() | Insert every element in the set of matched elements to the end of the target. |
| .prepend() | Insert content, specified by the parameter, to the beginning of each element in the set of matched elements. |
| .prependTo() | Insert every element in the set of matched elements to the beginning of the target. |

**Elemente umschliessen/wrappen:**

|  |  |
| --- | --- |
| .wrap() | Wrap an HTML structure around each element in the set of matched elements. |
| .wrapAll() | Wrap an HTML structure around all elements in the set of matched elements. |
| .wrapInner() | Wrap an HTML structure around the content of each element in the set of matched elements. |

**Elemente entfernen:**

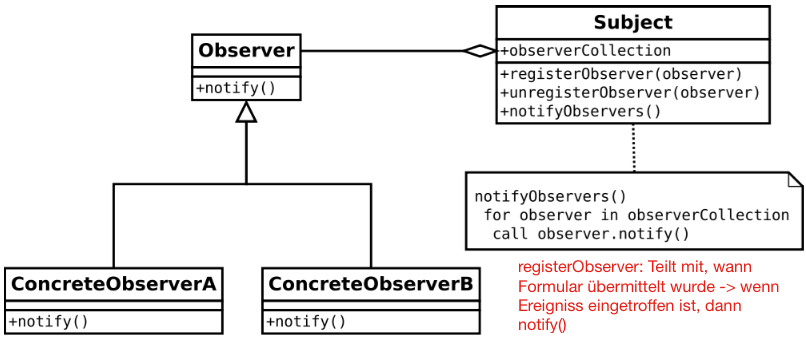
|  |  |
| --- | --- |
| .empty() | Remove all child nodes of the set of matched elements from the DOM. |
| .remove() | Remove the set of matched elements from the DOM, leaving the matched elements in their place. |
| → beide Methoden liefern noch das entfernte Element zurück | |
| .show() | Display the matched elements.  → ergänzen mit ‚slow’, ‚fast’… |
| .hide() | Hide the matched elements. |
| .toggle() | Display or hide the matched elements. |

# Events

**HTML Events**: Beispiel: <button onclick = 'getElementById("demo").innerHTML = Date() '>The time is?</button>

|  |  |
| --- | --- |
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

**Observer Pattern:**



**jQuery Events: Event handlers** oder **listeners:** Events können durch das System generiert werden (bspw. timer) oder können die Reaktion auf eine User-Eingabe sein (Maus, Keyboard, Touch).

**ready()-Event**: JS wird erst ausgeführt, wenn das Dokument vollständig geladen ist.

$(document).ready(function(){  
//jQuery methods go here... });

**Event-Methode:** Bspw. verstecken aller p-Elemente bei einem Doppelklick-Event.

**on()-Event** → .on(events, handler(eventObject))

$("p").on("click", function(){ $(this).hide(); });

**Spezifische** **Events**: → .eventypeName(listener)

$("p").dblclick(function(){ $(this).hide(); });

|  |  |  |  |
| --- | --- | --- | --- |
| blur | keypress, keydown | change | keyup click |
| load | click, dblclick | scroll | error |
| mousemove, mouseup, mousedown, mouseout | focus |  | resize |
| submit | unload |  | select |

**Delegated Event:** Können Elemente behandeln, welche erst zu einem späteren Zeitpunkt entstehen.

$("#dataTable tbody").on("click", "tr", function(event) });

**Element Targets und Delegate Targets:**

* event.currentTarget ist ormalerweise === this, also das DOM-Element welches den Event ausgelöst hat.
* event.delegateTarget ist das DOM-Element an welches der Event-Handler angehängt ist. Falls nicht delegiert, dann === currentTarget.
* event.target ist das DOM-Element welche den Event ausgelöst hat und normalerweise === currentTarget oder ein Child-Element davon.

preventDefault()

stopPropagation()

# Protocols

**TCP/IP (Transmission Control Protocol / Internet Protocol):** End-to-end-connectivety, spezifiziert wie Daten formatiert, adressiert, übermittelt, geroutet und empfangen werden. Besitzt 4 Abstraction Layer (hier vom untersten zum obersten):

* Link Layer: enthält Kommunikationstechnologien für ein einzelnes Netzwerk
* Internet Layer: verbindet unabhängige Netzwerke
* Transport Layer: behandelt Host-to-Host-Kommunikation
* Application Layer: Enthält alle Protokolle für spezifische Kommunikation auf Process-to-process-Level

**DNS (Domain Name System):** Verbindet Domain Names mit Entitäten, übersetzt Domain Names in numerische IP-Adressen (IPv4 und IPv6)

**URL (Uniform Ressource Locator):** auch „Web-Adresse“, ist ein String der eine Referenz zu einer Ressource macht. Bsp. http(Protokoll)://(„://“)en.example.org(Host)/wiki/Main\_Page(vollständige Adresse)

**HTTP (Hypertext Transfer Protocol):** ist Fundament des WWW, Protokoll zum Transfer und Austausch von Hypertext. Wichtigster Vorteil gegenüber anderen Protokollen ist Skalierbarkeit.

→ Client setzt HTTP Request an Server ab, dieser retourniert eine Response Message (Inhalt: Completion Status Information) zum Client. **HTTP Request-Methoden:**

* POST, GET, HEAD, PUT, DELETE, OPTIONS, CONNECT, PATCH
* HTTP-Status Codes: status line (=erste Zeile einer Response) besteht aus einem status code und einer reason phrase → Bspw. *404 Not found* für „angeforderte Ressource nicht gefunden“

**REST → vgl. Kapitel REST**

**XHR (XML HTTP Requests):** verwendet um HTTP/HTTPS Requests an einen Server zu senden und Response-Data (als JSON, XML, HTML o. plain text) zurück ins Script zu laden. Data kann Page verändern ohne diese neu laden zu müssen. XHR ist wichtig bei Ajax. Methoden sind die gleichen wie bei HTTP Responses.

**Ajax → vgl. Kapitel Ajax**

**JSON (JavaScript Object Notation):** ist ein textbasierter open standard, abgeleitet von JavaScript (ist aber Sprach-unabhängig). Wird oft verwendet zur Serialisierung und Übermittliung von strukturierten Daten über ein Netzwerk → hauptsächlich zwischen Server und Web API. Typen: String, Number, Boolean, Array, Object, null. Bsp.:

{"employees":[ //JSON-Objekt mit 3 Angestellten

{"firstName":"John", "lastName":"Doe"},

{"firstName":"Anna", "lastName":"Smith"},

    {"firstName":"Peter", "lastName":"Jones"} ]}

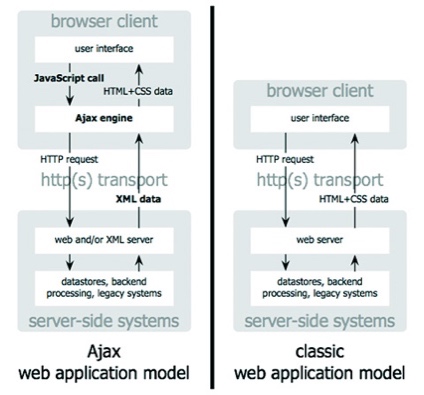
String wieder in ein JS-Object umwandeln mit parse():

var obj = JSON.parse(employees);

# Ajax (Asynchronous JavaScript and XML)

* WebAPIs können Data senden/empfangen von/zu einem Server → schnelle und dynamische Webseiten
* Requests sind asynchron ohne die Darstellung/das Verhalten der Page zu ändern. Requests welche den Status der Page ändern, tun dies via Events.

Besteht aus einer **Gruppe von Technologien** (XML nur historisch):

* HTML/XHTML and CSS can is used in combination to mark up and style information
* The DOM is accessed with JavaScript to dynamically display results
* JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server
* JSON, XML or plain text are used to exchange data

|  |
| --- |
| 1. XMLHttpRequest object erstellen: xhttp=new XMLHttpRequest(); 2. Request an Server senden:   open(method, url, async): Specifies the type of request  method: the type of request: GET (=default) or POST  url: the server (file) location  async: true (asynchronous) or false (synchronous)  send(): Sends request to server (used for GET)  send(string): Sends request to server (used for POST)   1. Server response:   responseText: get the response data as a string  responseXML: get the response data as XML data |

**Getting Data with jQuery**: $.get(url, parameters, callback)

**Getting JSON-Data:** $.getJSON(url, parameters, callback)

Making **POST-Request:** $.post(url, parameters, callback)

→ falls Callback nicht spezifiziert: GET als Default

**Ajax Request with full control:** $.ajax(options): Methoden:

type: The HTTP method to use. Usually either POST or GET. If omitted, the default is GET.

data: An object whose properties serve as the query parameters to be passed to the request. If the request is a GET, this data is passed as the query string. If a POST, the data is passed as the request body. In either case, the encoding of the values is handled by the $.ajax() utility function.

dataType: A keyword that identifies the type of data that’s expected to be returned by the response. This value determines what, if any, post-processing occurs upon the data before being passed to callback functions. The valid values are as follows: xml, html, json, jsonp, script, text. The server resource is responsible for setting the appropriate content-type response header. If this property is omitted, the response text is passed to the callbacks without any processing or evaluation.

timeout: Sets a timeout for the Ajax request in milliseconds. If the request does not complete before the timeout expires, the request is aborted and the error callback (if defined) is called.

global: Enables (if true) or disables (if false) the triggering of so-called global functions. These are functions that can be attached to elements that trigger at various points or conditions during an Ajax call. If omitted, the default is to enable the triggering of global functions.

contentType: The content type to be specified on the request. If omitted, the default is application/x-www-form-urlencoded, the same type used as the default for form submissions.

success: A function invoked if the response to the request indicates a success status code. The response body is returned as the first parameter to this function and formatted according to the specification of the dataType property. The second parameter is a string containing a status value—in this case, always success.

error: A function invoked if the response to the request returns an error status code. Three arguments are passed to this function: the XHR instance, a status message string (in this case, always error) and an optional exception object returned from the XHR instance.

complete: A function called upon completion of the request. Two arguments are passed: the XHR instance and a status message string of either success or error. If either a success or error callback is also specified, this function is invoked after the callback is called.

beforeSend: A function invoked prior to initiating the request. This function is passed the XHR instance and can be used to set custom headers or to perform other pre-request operations.

async: If specified as false, the request is submitted as a synchronous request, By default, the request is asynchronous.

processData: If set to false, prevents the data passed from being processed into URL- encoded format. By default, the data is URL-encoded into a format suitable for use with requests of type application/x-www-form-urlencoded.

ifModified: If true, allows a request to succeed only if the response content has not changed since the last request according to the Last-Modified header. If omitted, no header check is performed.

# REST (Representational State Transfer)

REST ist ein Architektur-Stil zum Designen von Netzwerk-API (Facebook, Twitter etc.). RESTful APIs verwendet HTTP-Requests um Daten zu übertragen, updaten etc. Ist ein Mechanismus um Daten zwischen Services zu transportieren unter Verwendung von HTTP Requests.

* request methods: GET, POST, PUT, PATCH, DELETE u. a.

**RESTful applications** benutzen HTTP- und REST-Prinzipien. HTTP Requests für post (create and/or update), read (bspw. make queries) und delete Data.

Stateless: Stateless Protocol (bspw. HTTP): Ist Eigenschaft, dass jeder Request isoliert von anderen geschieht → nie abhängig von anderen.

|  |  |
| --- | --- |
| HTTP method | Description |
| GET | List the URIs and perhaps other details of the collection's members. |
| PUT / PATCH | Replace the entire collection with another collection. |
| POST | Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation. |
| DELETE | Delete the entire collection |

# Promis

Ist ein Versprechen für etwas, das in der Zukunft passieren wird. Status kann nur 1x verändert werden → von pending (Initialisierungsstatus) zu fulfilled oder rejected.

import 'babelify/polyfill'

function loadImage(url) {return new Promise((resolve, reject)=>{

let image = new Image();

image.onload = function() {

resolve(image);

};

image.onerror = function() {

let message = 'Could not load image at ' + url;

reject(new Error(msg))

};

image.src = url;

})

};

export default loadImage;

# LocalStorage

**Cookies**: Wurden früh erfunden, haben aber 3 Nachteile (werden in jedem HttpRequest angehängt→verlangsamung, senden Daten unverschlüsselt über Web, limitiert auf 4 KB).

**Session Storage**: Data ist nur in Page von der gleichen Domain zugänglich, zugang über sessionStorage global object, geht verloren nach Sessions-Ende, wird erstellt bei jedem Öffnen eines Tabs/Windows.

**Local Storage**: Kann data für mehr als eine Session speichern, zugänglich über localStorage global object → funktioniert gleich wie sessionStorage, Daten bleiben aber gespeichert wenn Session beendet wird. Werden mit anderen Windows/Tabs geteilt.

* limitiert auf 5MB (sonst QUOTA\_EXCEEDED\_ERR)

Strage/Ablage basiert auf key/value-Pairs, wobei key ein String und value ein beliebiges Format ist. Nur 4 Funktionen:

* set value:

localStorage.setItem("key", value);

localStorage["key"] = value; //Array-Notation

* get value:

var value = localStorage.getItem("key");

var value = localStorage["key"]; //Array-Notation

→ weil Data nur als String vorhanden ist, muss bei Infos ≠ String parseInt() / parseFloat() o. ä. verwendet werden.

* remove value: localStorage.removeItem(„key“);
* clear all values: localStorage.clear(); // entfernt ALLE values

**Objekte holen** mit JSON.stringify() und JSON.parse():

localStorage.setItem( „key“, JSON.stringify(object) );

console.log( JSON.parse( localStorage.getItem( „key“ ) ) )

# Canvas

Markup: <canvas width="300" height="225"></canvas>

Get Context: $('canvas#a').click(function() { var context = this.getContext("2d"); });

Position: moveTo(x, y);

Fix lines: lineTo(x, y);

Begin Path: beginPath(); resets the current Path

Style: context.fillStyle /.strokeStyle = ‚#eee’;

Path zeichenen: stroke();

Text: context.font = ‚arial 12px sans-serif’; context.fillText(‚text’, x, y);

|  |
| --- |
| Bsp. Sinusfunktion: //Format y = amplitude \* sin(2π/period \* x)  function drawSinFunction(period, amplitude, color) {  context.beginPath();  for(x = 0; x <= maxX; x += 0.1) {  var y = amplitude \* Math.sin((2\*Math.PI/period) \* x);  context.lineTo(startX + x, startY - y);  };  context.strokeStyle = color;  context.stroke();  } |