JavaScript Fundamentals

Agenda

JavaScript Basics

Using DOM and Events

Introduction to Object Orientation

Object Orientation in Depth

Introduction to jQuery

Perform HTTP requests

ECMA Script 6

JavaScript Basics

What is JavaScript

JavaScript is a programming language designed for Web pages.

JavaScript enhances Web pages with dynamic and interactive features.

JavaScript runs in client software.

Unlike HTML, JavaScript is case sensitive

Syntax Overview

The JavaScript syntax is similar to C# and Java

- Operators (+, *, =, !=, &&, ++, …)
- Variables (typeless)
- Conditional statements (if, else)
- Loops (for, while)
- Arrays (my_array[]) and associative arrays (my_array['abc'])
- Functions (can return value)
- Function variables (like the C# delegates)

Implementing JavaScript

The JavaScript code can be placed in:

- <script> tag in the head
- <script> tag in the body
- External files, linked via <script> tag the head
  - Files usually have .js extension
  - The .js files get cached by the browser
  - Files can be minified to increase load time
- Inplace with the Tag using onclick event ...
- Use console.log insted of alert method

Implementing JavaScript Cont.

```
<!DOCTYPE html>
```

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8"/>
    <script type="text/javascript">
      function test (message) {console.log(message);}
    </script>
</head>
<body>
    <img src="logo.gif" onclick="test('clicked!')" />
</body>
</html>
```

```
ton");}}
```

```
<head>
    <title></title>
    <meta charset="utf-8"/>
    <script src="sample.js" type="text/javascript"/>
</head>
...
```

Execution

JavaScript code is executed during the page loading or when the browser fires an event

- All statements are executed at page loading
- Some statements just define functions that can be called later

Function calls or methods can be attached as callbacks via attributes or by code

- Executed when the event is fired by the browser

```
var el = document.getElementById("btnSave");
el.onclick = function() {
    //Do Save
};
```

Debugging JavaScript

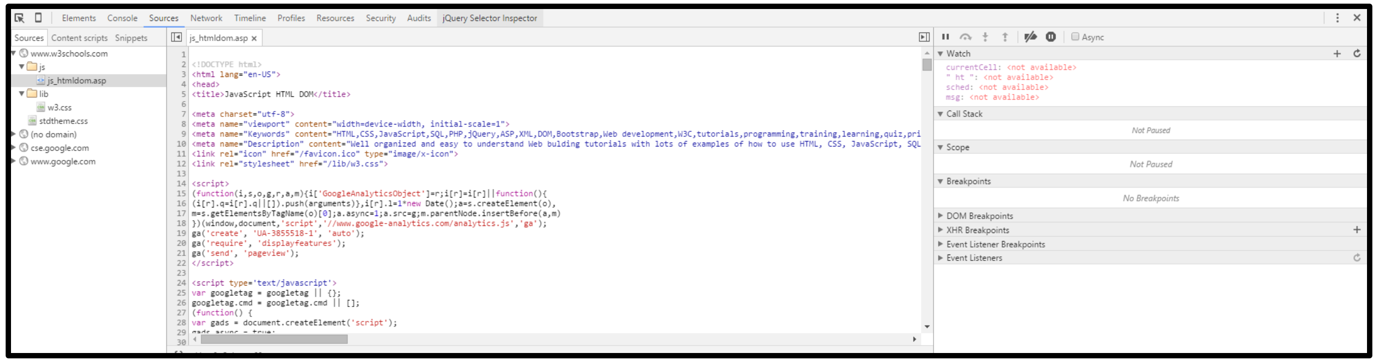
Modern browsers have JavaScript console where errors in scripts are reported

- Errors may differ across browsers

Several tools to debug JavaScript

- IE, Chrome, Firefox Developer Tools
  - Supports breakpoints, watches
  - JavaScript statement debugger; opens the script editor

Many useful Add-Ons for Firefox and Chrome available



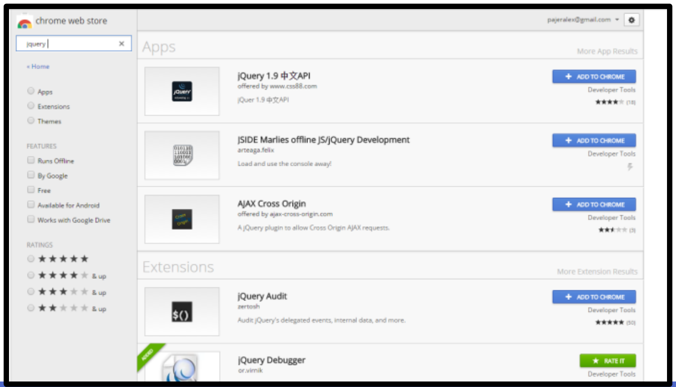
```
<!DOCTYPE html>
<html lang="en-US">
<head>
<title>JavaScript HTML DOM</title>

<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="Keywords" content="HTML,CSS,JavaScript,SQL,PHP,jQuery,ASP,XML,DOM,Bootstrap,Web development,W3C,tutorials,programming,training,learning,quiz,pri
<meta name="Description" content="Well organized and easy to understand Web bulding tutorials with lots of examples of how to use HTML, CSS, JavaScript, SQL
<link rel="icon" href="/favicon.ico" type="image/x-icon">
<link rel="stylesheet" href="/lib/w3.css">

<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','//www.google-analytics.com/analytics.js','ga');
ga('create', 'UA-3855518-1', 'auto');
ga('require', 'displayfeatures');
ga('send', 'pageview');
</script>

<script type='text/javascript'>
var googletag = googletag || {};
googletag.cmd = googletag.cmd || [];
(function() {
var gads = document.createElement('script');
```

```
currentCell: <not available>
" ht ": <not available>
sched: <not available>
msg: <not available>
```



jQuery 1.9 中文API
offered by www.css88.com
jQuer 1.9 中文API

JSIDE Marlies offline JS/jQuery Development
arteaga.felix
Load and use the console away!

AJAX Cross Origin
offered by ajax-cross-origin.com
A jQuery plugin to allow Cross Origin AJAX requests.

jQuery Audit
zertosh
Audit jQuery's delegated events, internal data, and more.

jQuery Debugger
or.vimk

Console Object

The console object exists only if there is a debugging tool that supports it

- Used to write log messages at runtime

Methods of the console object:

- debug(message)
- info(message)
- log(message)
- warn(message)
- error(message)

Data Types

Primary Data Types

- String
- Number
- Boolean

Composite Data Types

- Object
- Array

Special Data Types

- Null
- Undefined

Variables & Types

Variables are defindes using var keyword

No explicit types in JavaScript

JavaScript supports Type Conversion

You can use the typeof operator to find the data type of a JavaScript variable.

```
var string1 = "fat ";
var x = 9.656;
x.toFixed(2);             // returns 9.66
parseInt("10.33");        // returns 10

String(123)
(123).toString()
typeof "John“ //string
typeof {name:'John', age:34} //object
```

Value (Primitive) / Reference Types

In JavaScript there are 5 primitive types:

- boolean, string and number
- undefined, null,

When primitive Types are passed into a function a copy of their value is passed

When object Types are passed into a function a pointer to the object is passed

Compare ByRef / ByVal in C#

Global Namespace

The concept of namespaces does not exist in JavaScript.

To add insult to injury, everything you create in JavaScript is by default global

What would happen if they use the same names? Well, if they are includedafter our script, they will overwrite our variables and function

To solve this problem you can create a single global object for your app and make all functions and variables properties of that global object

Scoping

Variables declared within a JavaScript function, become LOCAL to the function

A variable declared outside a function, becomes GLOBAL

If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable

```
var x = 10;

function math() {
    var pi = 3.14;
}

console.log(x);
console.log(math.pi);
```

Arrays

Arrays are create using Array object or by assingment

Support basic methods like

- Push / Pop
- Sort
- ForEach
- Lenght

Arrays

Declaring new empty array:

```
var arr = new Array();
```

Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

Appending an element / getting the last element:

```
arr.push(3);
var element = arr.pop();
```

Reading the number of elements (array length):

```
var nbr = arr.length;
```

Basic JavaScript

DEMOS -> REFERENCING, EXECUTING

DEMOS -> DEBUGGING & LOGGING

DEMOS -> ARRAYS

Dates

Date

The Date object lets you work with dates (years, months, days, hours, minutes, seconds, and milliseconds)

Date objects are created with the overloaded **new Date()** constructor.

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Moment.JS

A JS lib for date / time manipulation

Available at http://momentjs.com/

Install using bower install moment –save

Docu published at http://momentjs.com/docs/

```
console.log("Using moments.js with Dates");
console.log("Date format: " + moment().format('MMMM Do YYYY, h:mm:ss a'));
console.log("Relative time: " + moment("20111031", "YYYYMMDD"));
console.log("Add time: " + moment().add(1, 'days'));
console.log(moment("13-02-99", "DD-MM-YY").format('YYYY-MM-DD'));
console.log(moment("13.02.2016", "DD-MM-YYYY").format('YYYY-MM-DD'));
```

Program Flow

Program Flow

single-selection structure (**if**),

double-selection structure (**if/else**),

inline ternary operator **?:**

multiple-selection structure (**switch**)

the expression is tested at the top of the loop (**while**),

the expression is tested at the bottom of the loop (**do/while**),

operate on each of an object's properties (**for/in**).

counter controlled repetition (**for**).

Conditional Statement – IF / Switch

Conditional statements are used to perform different actions based on different conditions.

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

```
switch (new Date().getDay()) {
    case 0: day = "Sunday"; break;
    case 1: day = "Monday"; break;
    case 2: day = "Tuesday"; break;
    case 3: day = "Wednesday"; break;
    case 4: day = "Thursday"; break;
    case 5: day = "Friday"; break;
    case 6: day = "Saturday"; break;
}
```

Loops

Like in C#, Java

- for loop
- for … in
- While loop
- Do … while loop

```
var counter;
for (counter = 0; counter < 4; counter++) {
    alert(counter);
}
var obj = { a: 1, b: 2, c: 3 };
for (var prop in obj) {
    console.log("obj." + prop + " = " + obj[prop]);
}
while (counter < 5) {
    alert(++counter);
}
```

Functions

Code structure – splitting code into parts

Data comes in, processed, result returned

Functions are not required to return a value

```
function average(a, b, c) {
    var total;
    total = a + b + c;
    return total / 3;
}
```

Parameters come in here.

Declaring variables is optional. Type is never declared.

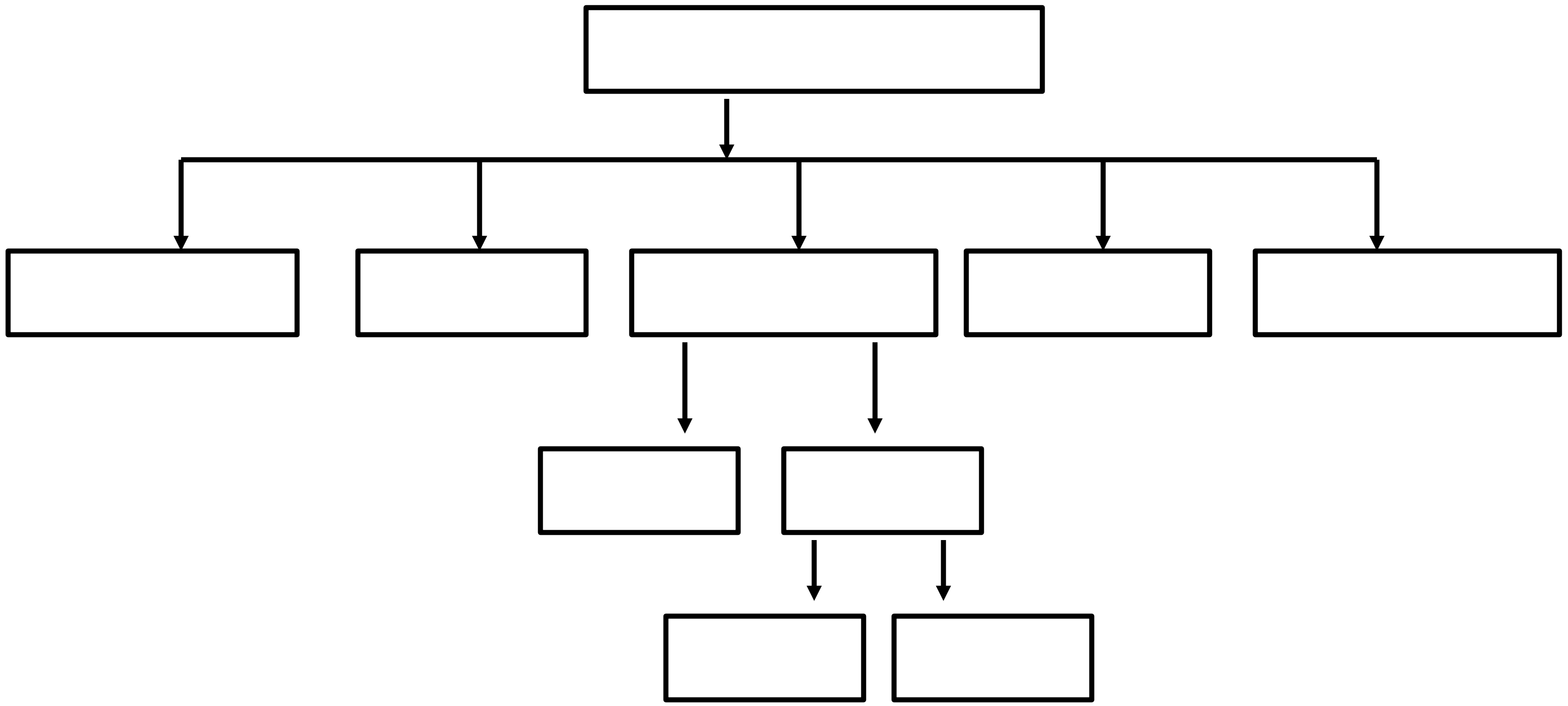
Value returned here.

Program Flow

DEMOS -> PROGRAM FLOW

Using DOM and Events

Browser Objects



Document Object Model

Every HTML element is accessible via the JavaScript DOM API

Most DOM objects can be manipulated by the programmer

The event model lets a document to react when the user does something on the page

Advantages

- Create interactive pages
- Updates the objects of a page without reloading it

Document Object Model

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

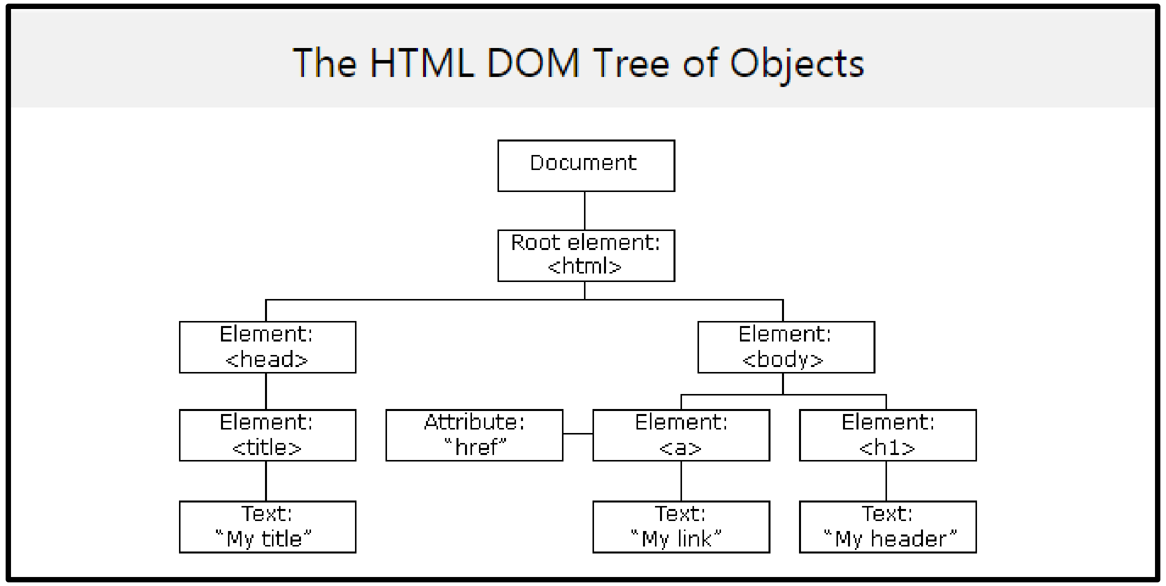
The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

The HTML elements as **objects**

The **properties** of all HTML elements

The **methods** to access all HTML elements

The **events** for all HTML elements



Using DOM

Access elements

- via their ID attribute
- Via the name attribute
- Via tag name

Once we access an element, we can read and write its attributes

```
function change(state) {
    var lampImg = document.getElementById("lamp");
    lampImg.src = "lamp_" + state + ".png";
    var statusDiv =
      document.getElementById("statusDiv");
    statusDiv.innerHTML = "The lamp is " + state;
}
```

DOMContentLoaded

The DOMContentLoaded event is fired when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading

Supported by IE 9+

```
<script>
  document.addEventListener("DOMContentLoaded", function(event) {
    console.log("DOM fully loaded and parsed");
  });
</script>
```

DOMContentLoaded vs window.onload

The DOMContentLoaded event is fired when the document has been completely loaded and parsed, without waiting for

- stylesheets,
- images, and
- subframes

to finish loading

The load event can be used to detect a fully-loaded page

Sequence

- Scripts will be loaded
- DomContentLoaded will be fired
- window.onload

Events

All event handlers receive one parameter

- It brings information about the event
- Contains the type of the event (mouse click, key press, etc.)
- Data about the location where the event has been fired (e.g. mouse coordinates)
- "this" holds a reference to the event sender
    - E.g. the button that was clicked

Common Events

Mouse events:

- onclick, onmousedown, onmouseup
- onmouseover, onmouseout, onmousemove

Key events:

- onkeypress, onkeydown, onkeyup
- Only for input fields

Interface events:

- onblur, onfocus
- onscroll

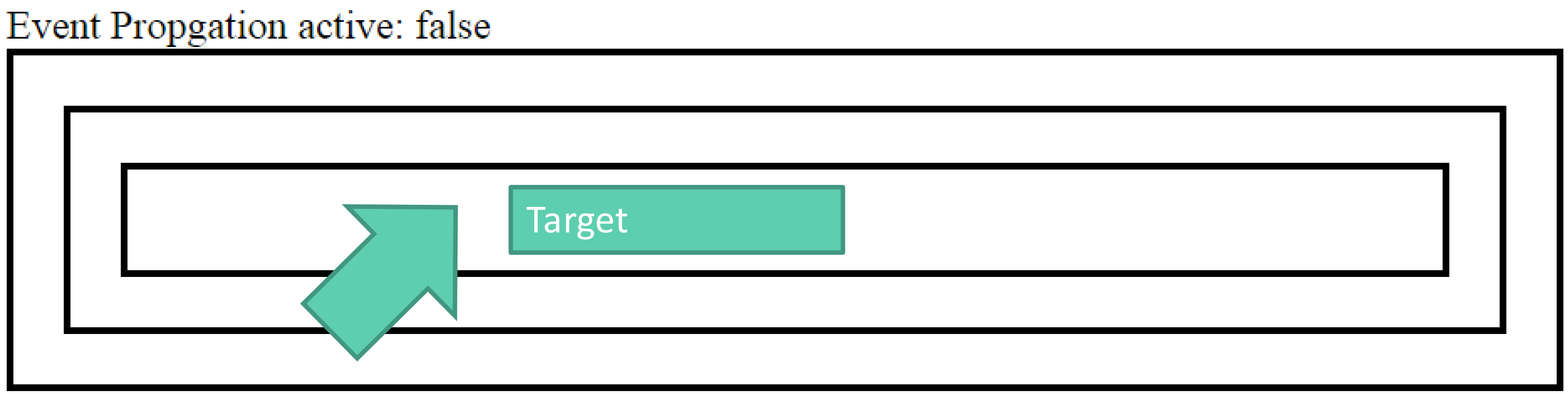
Form events

- onchange – for input fields
- onsubmit
  - Allows you to cancel a form submission
  - Useful for form validation

Event Bubbling

By default events are passed up from on element to its parent element

Can be turned off using: event.stopPropagation()



Object Oriented JavaScript

Introduction

The simplest way is to use the built-in Object data type

In JavaScript, objects are implemented as a collection of named properties

JavaScript allows the creation of any number of properties in an object at any time

Object-oriented paradigms.

- Encapsulation
- Inheritance

```
var student = new Object;
student.name = "Homer Simpson";
student.grade = 3;

var obj = {}
obj.NAME = "Hans";
obj.methode = function () {
        console.log("Modul-Eigenschaft: " + obj.NAME);
        };

obj.methode();
```

Object Initializer

An object initializer is perhaps the simplest way to encapsulate related code

Doesn't offer any privacy for properties or methods

It's useful for eliminating anonymous functions from your code

```
var dog = {
    name: "Giro",
    bark: function () {
        console.log("My name is: " + this.name);
    },
    hunt: function () {
        console.log("I am hunting a rabbit");
    }
}
console.log(dog.name);
dog.bark();
```

Constructor Funtions

A new JavaScript class is defined by creating a function (serving as constructor)

- When used with the `new` operator, a function serves as a constructor for that class
- Internally, JavaScript creates an `Object`, and then calls the constructor function
- When defining a constructor function, we can make many objects with the same properties

```
function soccerPlayer(name, goals) {
    this.name = name;
    this.goals = goals;
}
var arn = new soccerPlayer("Marco Arnautovic", 3);
var alb = new soccerPlayer("David Alaba", 2);
var jan = new soccerPlayer("Marc Janko", 4);
```

Constructor Function with method

We can add a methods to the object (instance) at any time

```
function SoccerPlayer(name, goals) {
    this.name = name;
    this.goals = goals;
    this.scoreGoal = function () {
        console.log("I am " + this.name + " and I scored " + this.goals + " goals!");
    }
}
var marko = new SoccerPlayer("Marko Arnautovic", 3);
marko.tellScore = function () { console.log("Until now I scored " + this.goals + " goals") }
marko.scoreGoal();
marko.tellScore();
```

Using this in constructor functions

When used in a function this always points to the containing object of the function

Properties created using this are public by default

```
var Person = function(name) {
    var Name = name;              //not accessible from outside
    this.FirstName = name;
    this.sayName = function() {
        console.log(Name);
        console.log(this.Name);      //undefined
        console.log(this.FirstName);
    }
}
var hugo = new Person("hugo");
hugo.sayName();
console.log(hugo.FirstName);
console.log(hugo.Name);
```

Prototypes

We can use the prototype object to add custom properties / methods to objects

Reflected on all instances of the class

```
function Circle(radius) {
    this.Radius = radius;
}

Circle.prototype.pi = Math.PI;
Circle.prototype.calculateArea = function () {
    console.log("Area of a cirle with radius " + this.Radius + " is: " + this.pi *
    Math.pow(this.Radius, 2));
}

var circleA = new Circle(10);
circleA.calculateArea();
```

Prototype Functions to existing Classes

Dynamically add a function to a built-in class at runtime using the prototype object

Use like C# Extension Methods

```
Array.prototype.showMax =
  function () {
      var max = this[0];
      for (var i = 1; i < this.length; i++) {
          if (max < this[i]) {
              max = this[i];
          }
      }
      return max;
  }
var array = new Array(9, 1, 11, 3, 4);
var max = array.showMax();
Console.log(max);  // 11
```

Attaching a method to the Array class

Inheritance

To inherit a class in JavaScript set the prototype object of the subclass to the superclass class

To execute the constructor of the supercalss use SUPERCLASS.call(this, props, ...)

```
function Person(name) {
    this.name = name;
    this.sayName = function() {
        console.log("Hi, I am " + this.name);
    }
}

function Student(name, grade) {
    Person.call(this, name);
    this.grade = grade;
}
Student.prototype = new Person();

var alex = new Student("Alex", "First");
alex.sayName();
```

This way we say that the Student class will have all the functionality of the Person class

Method overriding

Methods can be overridden using prototype

```
function Student(name, grade) {
    debugger;
    Person.call(this, name);
    this.grade = grade;
}

Student.prototype = new Person();
Student.prototype.sayName = function () {
    console.log("Hi, I am " + this.name + ", i go to grade " + this.grade);
}
```

Object Orientation in Depth

Encapsulation - Immediately-Invoked Function Expression (IIFE)

A JavaScript design pattern which produces a lexical scope using JavaScript's function scoping.

Can be used to avoid variable hoisting from within blocks, protect against polluting the global environment

Allow public access to methods while retaining privacy for variables defined within the function

```
(function () {
// the code here is executed once in its own scope
})();

(function (a, b) {
    // a == 'hello'
    // b == 'world'
})('hello', 'world');
```

IIFE – Private Variables and Accessors

IIFEs are also useful for establishing private methods for accessible functions while still exposing some properties for later use

```
(function () {
    var Person = {
        name: null,
        init: function (personName) {
            this.name = personName;
        },
        sayName: function () {
            console.log("Hi, i am " + this.name);
        }
    }

    Person.init("Josef");
    Person.sayName();
})();
```

Module Pattern

Expose properties and methods using return {....}

```
var worker = (function (personName) {
    var name = personName;
    var wealth = 0;
    console.log("A new worker was born!");
    console.log("I am " + name + " & my current balance is " + wealth);

    function writeBalance() {
        console.log("my current balance equals " + wealth);
    }
    return {
        workfor: function (amount) {
            wealth += amount;
            writeBalance();
        }
    };
})("Franz");
```

Namespaces

Namespaces can be simpulated using Object Initializers

```
//Namespaces
debugger;

var VoucherEditor = {};
VoucherEditor.Modul1 = (function () { console.log("Modul1 in Namespace VoucherEditor"); })();
VoucherEditor.Modul2 = (function () { console.log("Modul2 in Namespace VoucherEditor"); })();

//Execute
VoucherEditor.Modul1();
```

Introduction to jQuery

What is jQuery

Shortcut for jQuery is $

jQuery is a cross-browser JavaScript library

- Designed to simplify the client-side scripting of HTML
- The most popular JavaScript library in use today
- Free, open source software

jQuery's syntax is designed to make it easier to

- Navigate a document and select DOM elements
- Create animations
- Handle events
- Implement plugins
- Develop responsive applications

Benefits of jQuery

Easy to learn

- Fluent programming style

Easy to extend

- You create new jQuery plugins by creating new JavaScript functions

Powerful DOM Selection

- Powered by CSS 3.0

Lightweight

Community Support

- Large community of developers and geeks

Installing jQuery

jQuery 1.x with support for older IE Versions including IE 6, 7, 8

jQuery 2.x no support for older IE Versions

jquery-migrate-1.2.1.min - restores deprecated features and behaviors so that older code will still run properly on jQuery 1.9 and later.

Manually download from http://jquery.com/download/

Documentation @ https://jquery.com/

Install using bower

```
bower install jquery
bower install https://code.jquery.com/jquery-2.2.2.min.js
```

Using jQuery

With jQuery you typically find something, then do something with it

- Syntax for finding items is the same as the syntax used in CSS to apply styles
- There are lots of different jQuery methods to do with the selected elements

```
if (window.jQuery) { //check for jQuery
    // Finding the item
    var st = $("#something");
    // Doing something with the found item
    st.hide();      //Does not work
    $(st).hide();   //Works
}
```

$(document).ready(function()

$(document).ready() is executed after the page – the document is "ready"

Because this event occurs after the document is ready, it is a good place to have all other jQuery events and functions

Can be compared to PageLoad event in classical ASP.NET

```
$(document).ready(function ($) {
    //do jQuery stuff when DOM is ready
});
```

jQuery Selectors

Selectors are just like in CSS used to select one ore more elements (tags)

Most important selectors are:

- ID Selector: $("#elementID")
- Class Selector: $(".cssClassName")
- Element Selector: $("p")

Reference pulished at http://api.jquery.com/category/selectors/

- Any action taken will typically affect all the elements you have selected

```
<div class="myClass" />
<div class="baz myClass" />
<div class="bar" />
</div>
```

```
$(".myClass").hide();
```

jQuery collections

$('div.section') returns a jQuery collection - You can call treat it like an array

```
$('div.section').length;     // no. of matched elements
$('div.section')[0]      // the first div DOM element
$('div.section')[1]
$('div.section')[2]

$('div.section').each(function() {
    console.log(this);
});
```

Type Testing Functions

Determine the type of an object

Useful for optional parameters & validation

- $.isArray(array)
- $.isFunction(function)
- $.isEmptyObject(object)
- $.isPlainObject(object)
- $.isXmlDoc(doc)
- $.isNumeric(number)
- $.isWindow(window)
- $.type(object)

Manipulating HTML / CSS

Change values in existing tags

Manipulate attributes

Manipulate CSS

```
$('a.nav').attr('href', 'http://mxx.com/');
$('a.nav').attr({'href': 'http://mxx.com/', 'id': 'flickr'});
$('#intro').removeAttr('id');
```

```
$('#intro').addClass('highlighted');
$('#intro').removeClass('highlighted');
$('#intro').toggleClass('highlighted');
$('p').css('font-size', '20px');
$('p').css({ 'font-size': '20px', color: 'red' });
```

Adding / Removing Elements to DOM

You can also add / remove elements from the DOM

```
<div>
     <p>Red</p>
     <p>Green</p>
</div>
```

```
$('<ul><li>Hello</li></ul>').appendTo('body');
$('p').remove();
```

Binding Events

Use $(selector).evt(function(){….}) pattern to bind a single event

Use on / off to bind to multible elements at the same time

on can be bount to all elements in a container – even if they are added late on

```
$("#divLiveParent").on("click", "button", showTarget);
$("#nbrInput").on("blur", "input", showTarget);
$("#nbrInput").on("focus", "input", showTarget);
$("#divLiveParent").off('click');
```

Custom Events

jQuery allows you to implement custom events

Leads to a more event oriented pattern

```
$(".vdElement").on("resetForm", function () {
        $("#fullname").val("");
        $("#password").val("");
});

$("btnReset").trigger("resetForm");
```

Perform HTTP requests

jQuery Ajax

You can use jQuery Ajax to seamlessly integrate with server side functionality

- jQuery makes simple the asynchronous server calls
- The core method for using AJAX functionality
- The shortcut methods use it 'under the hood'
- More control over what you want to do
- Get result as XML or JSON

Detailed docu at http://api.jquery.com/jquery.ajax/

jQuery.ajax() Sample

```
$.ajax({
        type: "POST",
        data: JSON.stringify({ 'Data': param }),
        url: "/Services/MaintenanceService.svc/SetScheduleEntry",
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: function (msg) {
            ....
        },
        error: function () {
            alert('Fehler beim Speichern');
        }
    });
```

jQuery.ajax() - Type

Specifies the type of request.

Mostly GET (not sending any data - read) or

POST (sending data - create)

Other HTTP verbs usd when consuming RESTful services

- PUT - Update/Replace
- PATCH - Update/Modify
- DELETE - Delete

Other Methods

$.getJSON()

- Shortcut for $.ajax() ... Less control

$.get(…) and $.post(…)

- Executes a server-side request and returns a result
- The HTTP action that will occur is POST or GET
- $.getScript(..)
- Loads a script and executes it

Promises & Deferreds

Consists of two main elements

- A deferred represents work that is not yet finished
- A promise represents a value that is not yet known

3 states: unfulfilled, fulfilled and failed and may only move from unfulfilled to either fulfilled or failed

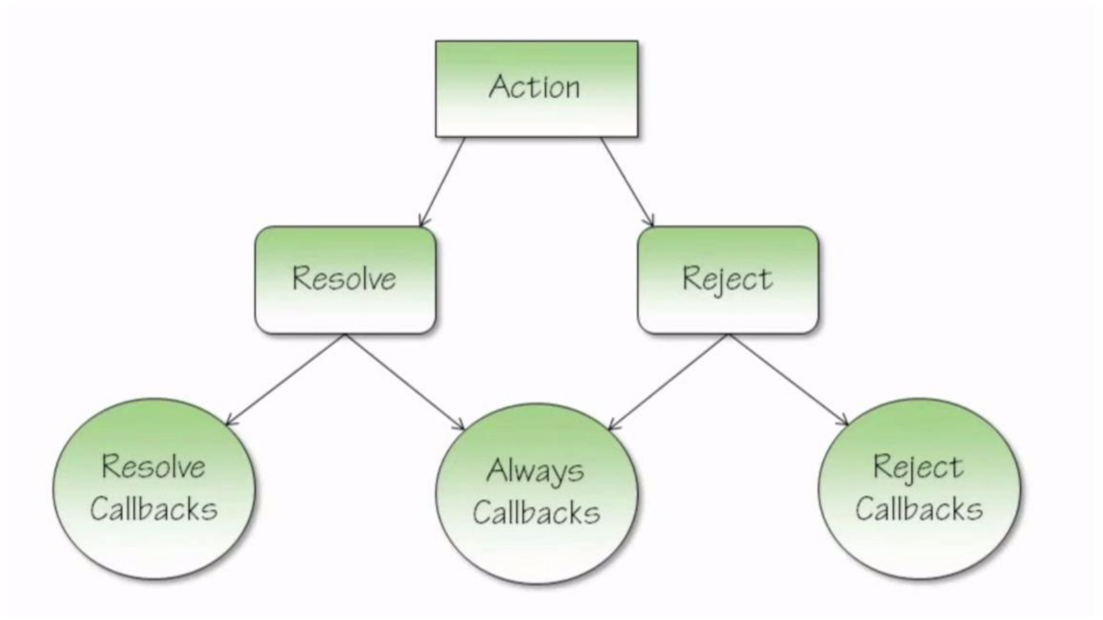
Considered from a high level, promises in JavaScript give us the ability to write asynchronous code in a parallel manner to synchronous code

Deferred

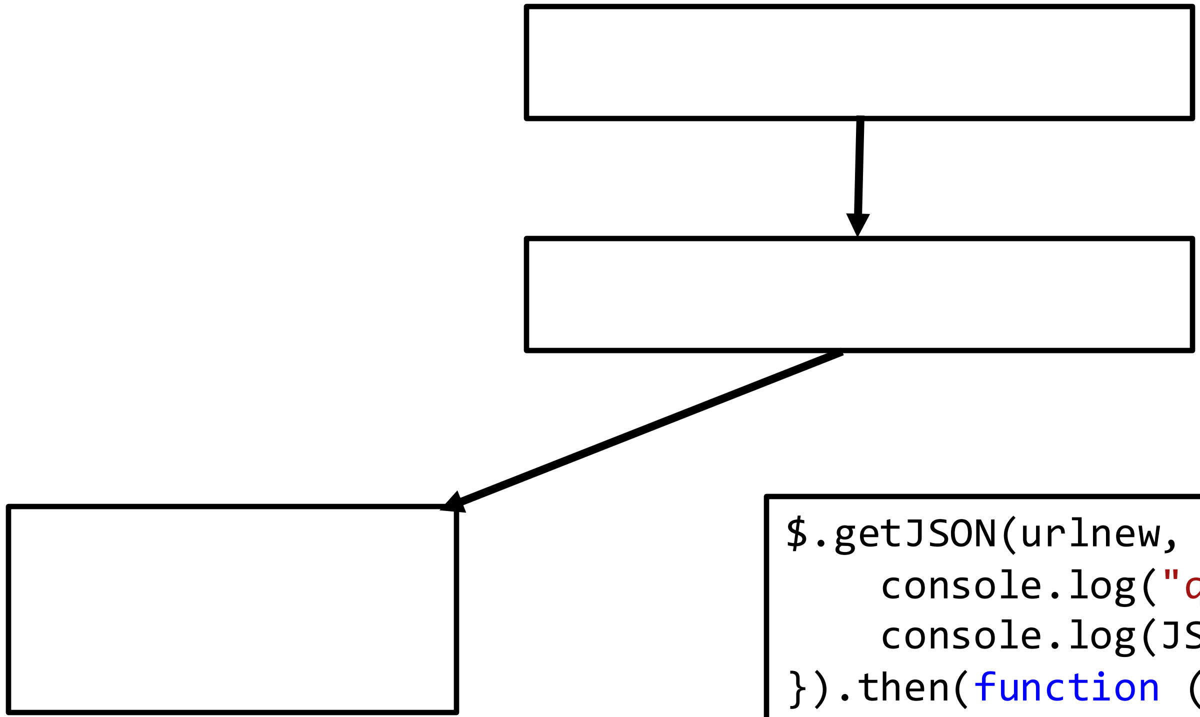
An object for holding & calling a queue of success / failure / complete callbacks

Advantage over typical callback registration

- Allows multibele callback registration
- Allows registration of callback at any time
    - Creation
    - Execution
    - After Comp0ltion



Deferred -> Promis



```
$.getJSON(urlnew, function (data) {
    console.log("query executed - response ok");
    console.log(JSON.stringify(data));
}).then(function (result) {
    console.log("Entering \".then-branch-success\"");
}).fail(function (error) {
    console.log("Entering \".fail-branch\"");
});
```

Promises

DEMO -> PROMISES

Callbacks and Async Programming

Classical Async Request

```
function loadXMLDoc() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == XMLHttpRequest.DONE ) {
            if(xmlhttp.status == 200){
                document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
            }
            else if(xmlhttp.status == 400) {              alert('There was an error 400')           }
            else {alert('something else other than 200 was returned')          }
        }
    }
    xmlhttp.open("GET", "ajax_info.txt", true);
    xmlhttp.send();
}
```

INTEGRATIONS

Async Request & Callback with jQuery

Allows further configuration

Documented on http://api.jquery.com/jquery.ajax/

```
$.ajax({
    url: "test.html",
    context: document.body,
    success: function(){
        $(this).addClass("done");
    },
    error: function(){
        console.log('error');
    }
});
```

Consuming Data using jQuery

VOUCHERSJQUERY -> DEMOS -> CONSUMINGSERVICES.HTML

What's new with ECMA Script 6

ECMA Script 6

Lates JavaScript Standard

Full list of features: http://es6-features.org/

Not supported by all browsers

Compatibility Table: http://kangax.github.io/compat-table/es6/

Variables

ES6 introduces new ways to declare variables:

- let – creates a scope variable
  - Accessible only in its scope
- const – creates a constant variable
  - Its value is read-only and cannot be changed

```
for(let number of [1, 2, 3, 4]){
    console.log(number);
}
//accessing number here throws exception

const MAX_VALUE = 16;
MAX_VALUE = 15; // throws exception
```

String Functions

Template Literals using Backticks `...` and ${*VARIABLE*}

String.prototype.repeat / String.prototype.contains

String.prototype.startsWith / String.prototype.endsWith

```
//Template Literals
var productID = 100;
var category = "music";
var url = "http://server/" + category + "/" + productID;
var templateLiteral = `http://server/${category}/${productID}`;

//startsWith
var str = 'To be, or not to be, that is the question.';
console.log(str.startsWith('To be'));       // true
console.log(str.endsWith('question.'));     // true
```

Default Parameters

Default function parameters allow formal parameters to be initialized with default values if no value or undefined is passed.

```
function multiply(a, b = 1) {
    return a*b;
}

multiply(5); // 5
```

Array

Array.prototype.find

Array.prototype.fill

Array.prototype.keys/entries => Array Iterator

```
var inventory = [
    {name: 'apples', quantity: 2},
    {name: 'bananas', quantity: 0},
    {name: 'cherries', quantity: 5}
];
function findCherries(fruit) {
    return fruit.name === 'cherries';
}
console.log(inventory.find(findCherries)); // { name: 'cherries', quantity: 5 }
```

For-of-loop

The for-of loop iterates over the values

- Of an array
- Of an iteratable object

```
var someArray = ["a", "b", "c"];
for (var item in someArray) {
    console.log(item); // 0,1,2 ... Returns the key ... the index
}

for (var item of someArray) {
    console.log(item); // a, b, c
}
```

Map

The Map object is a simple key/value map.

Any value (both objects and primitive values) may be used as either a key or a value.

```
var myMap = new Map();
var keyString = "a string",
    keyObj = {},
    keyFunc = function () { };
// setting the values
myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, "value associated with keyObj");
myMap.set(keyFunc(), "value associated with keyFunc");
console.log("Map size: " + myMap.size); // 3
// getting the values
myMap.get(keyString);    // "value associated with 'a string'"
myMap.get("a string");   // "value associated with 'a string'" because keyString === 'a string'
myMap.get(keyObj);       // "value associated with keyObj"
```

Sets

The Set object lets you store unique values of any type, whether primitive values or object references.

Can iterate its elements in insertion order. A value in the Set may only occur once; it is unique in the Set's collection.

```
var mySet = new Set();
mySet.add(1);
mySet.add("some text");
var o = { a: 1, b: 2 };
mySet.add(o);

mySet.has(1); // true
mySet.has(3); // false, 3 has not been added to the set
mySet.has(Math.sqrt(25));  // true
mySet.has("Some Text".toLowerCase()); // true
mySet.has(o); // true
mySet.size; // 4
mySet.delete(5); // removes 5 from the set
```

REST Parameter

...items

Allows calling a function with a variable numer of arguments without using the arguments object

```
store.add('fruit', 'apple');
store.add('dairy', 'milk', 'cheese', 'yoghurt');
store.add('pastries', 'donuts', 'croissants');

store.add = function(category, ...items) {
    items.forEach(function (item) {
        store.categoryname[category].push(item);
    });
};
```

Spread Operator

The **spread operator** allows an expression to be expanded in places where multiple arguments (for function calls) or multiple elements (for array literals) are expected

```
var a, b, c, d, e;
a = [1, 2, 3];
b = "dog";
c = [42, "cat"];

// Using the concat method.
d = a.concat(b, c);

// Using the spread operator.
e = [...a, b, ...c];

console.log(d);
console.log(e);

// Output:
// 1, 2, 3, "dog", 42, "cat"
// 1, 2, 3, "dog", 42, "cat"
```

Arrow Functions

Known als Lambda Functions in C#

```
numbers.sort(function(a, b){
    return b - a;
});
```

```
numbers.sort((a, b) => b - a);
```

```
var fullnames = people.filter(p => p.age >= 18).map(p => p.fullname);
```

INTEGRATIONS

Destructuring Arguments

Destructuring assignments allow to set values to objects in an easier way

```
var [a,b] = [1,2]; //a = 1, b = 2
var [x, , y] = [1, 2, 3] // x = 1, y = 3
var [first, second, ...rest] = people;
```

```
var person = {
     name: 'Jack Wolf',
     address: {
          city: 'Vienna',
          street: 'Kärtner Straße 1'
    }
};

var {name, address: {city}} = person;
```

Classes

ES6 introduces classes and a way to create classical OOP

```
class Person extends Mammal{
    constructor(fname, lname, age){
        super(age);
        this._fname = fname;
        this._lname = lname;
    }
    get fullname() {
        //getter property of fullname
    }
    set fullname(newfullname) {
        //setter property of fullname
    }
    // more class members…
}
```