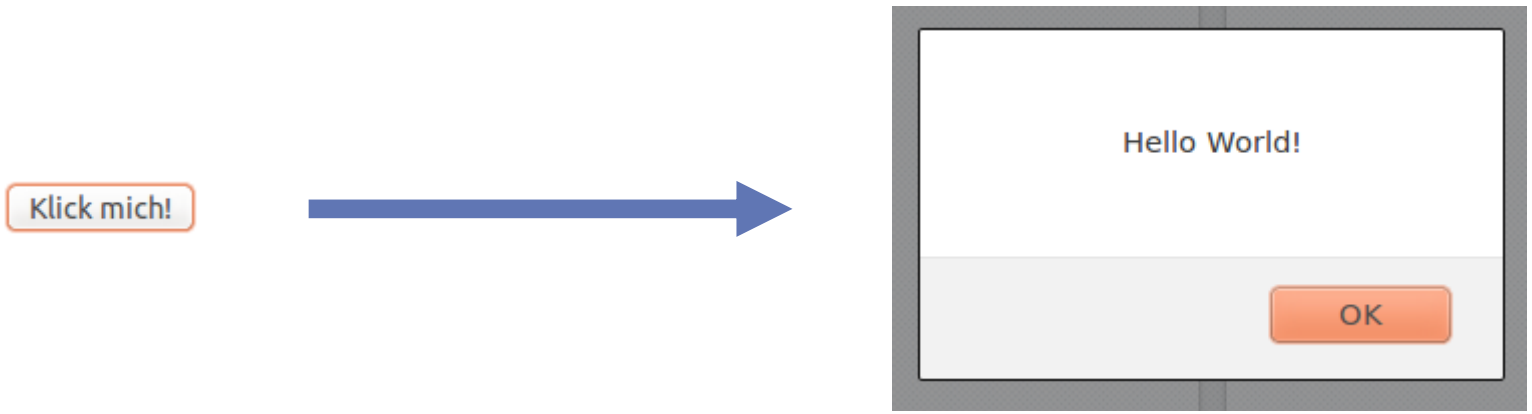
# Webapplikationen mit JavaScript

JavaScript

# JavaScript – Einführung

- JavaScript bildet mit HTML und CSS die Grundlage der meisten Webseiten
- JavaScript ist eine der verbreitetsten Programmiersprache

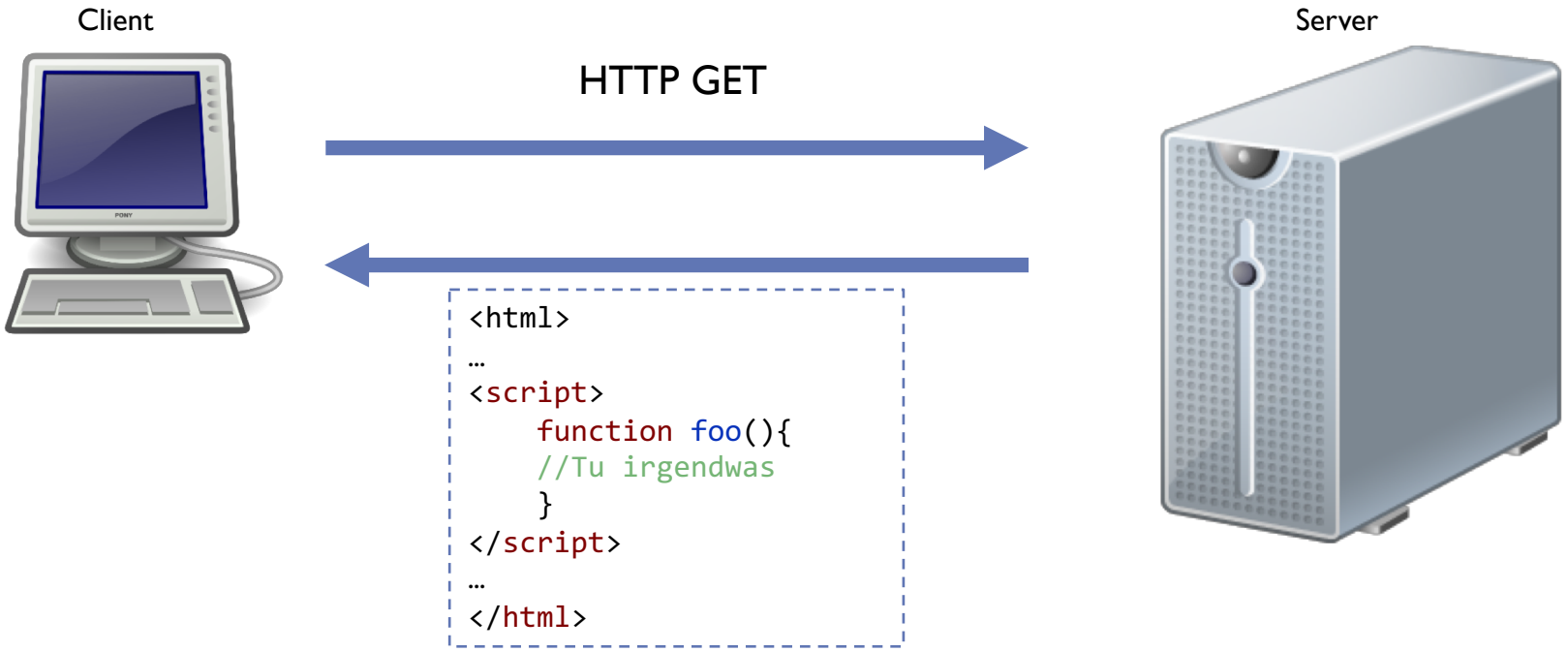


# JavaScript – Einführung

- Versionsgeschichte
  - JavaScript wurde 1995 von Netscape eingeführt und proprietär lizenziert
  - 1996 antwortete Microsoft mit JScript, einer eigenen Version von JavaScript, die über viele Windows-spezifische Erweiterungen verfügte
  - Beginn des sogenannten Browserkriegs
  - Netscape entwickelte in Kooperation mit dem ECMA (European Computer Manufacturers Association) die Spezifikation ECMAScript
  - 1998 wurde JavaScript Version 1.3 veröffentlicht, welche der ECMAScript-Spezifikation entsprach
  - Während des "Browserkrieges" mussten viele Webseiten auf beide Sprachen Rücksicht nehmen und doppelt programmiert werden. JavaScript entwickelte sich damals sehr rasch und erhielt viele neue Features um konkurrenzfähig zu bleiben
  - Die aktuelle Version ist ECMAScript 2018
  - Wir verwenden in den Übungen ECMAScript 5+ sowie TypeScript

# JavaScript – Einführung

- JavaScript hat mit Java (fast) nichts zu tun
- In JavaScript kann man objektorientiert, prozedural und funktional programmieren.



# JavaScript – Einführung

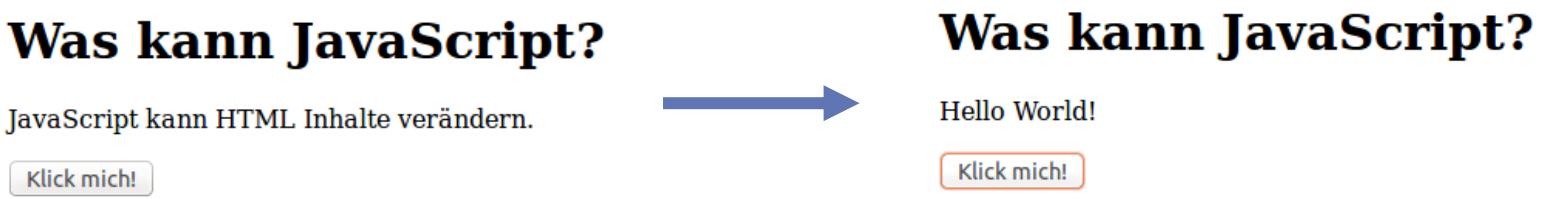
- Was kann JavaScript?
  - HTML-Inhalte ändern
  - HTML-Attribute modifizieren
  - CSS modifizieren
  - Eingaben validieren, etc.


  - JavaScript bildet die Grundlage vieler moderner Frameworks (Angular, Bootstrap, Node.js, …)

# JavaScript – Einführung

- Beispiel

```
<!DOCTYPE html>
<html>
<head>
<script>
    function myFunction() {
        document.getElementById("demo").innerHTML = "Hello World!";
    }
</script>
</head>
<body>
    <h1>Was kann JavaScript?</h1>
    <p id="demo">JavaScript kann HTML Inhalte verändern.</p>
    <button type="button" onclick="myFunction()">Klick mich!</button>
</body>
</html>
```



# JavaScript – Einbindung

- Einbindung im HTML-Dokument
  - JavaScript muss zwischen `<script>` und `</script>` platziert werden
  - Das Script kann sich im Head oder Body des HTML-Dokuments befinden

- Einbindung in externer Datei
  - Sollte die Dateiendung `.js` haben
  - In der Datei dürfen keine `<script>`–Elemente vorkommen
  - Einbindung in das HTML-Dokument mittels `<script src="myScript.js"></script>`
  - Das Script verhält sich dann so, als wäre es an der Stelle im HTML platziert worden
  - Externe Dateien für JavaScript sind in der Regel zu bevorzugen

# JavaScript – Syntax

- JavaScript-Syntax ähnelt der von Java
  - Einzelne Anweisungen müssen durch Semikolon getrennt werden
  - Anweisungsblöcke werden von oben nach unten abgearbeitet
  - JavaScript ist "Case Sensitive"

- JavaScript-Kommentare
  - Funktionieren wie in Java
  - Auskommentierter Text wird vom Browser ignoriert
  - Einzeilige Kommentare: `//Dies ist ein einzeiliger Kommentar`
  - Mehrzeilige Kommentare:
  - ```
    /*
     Dieser Kommentar erstreckt sich
     über mehrere Zeilen
    */
    ```
    ```

# JavaScript – Variable

## Variable deklarieren

- Schlüsselworte `var` (Funktionsscope) und `let` (Blockscope)
- `let student;` Erzeugt eine Variable namens "student" mit dem Wert `undefined`
- `student = 'Max';` weist dieser Variable den Wert "Max" zu
- Alternativ in nur einem Ausdruck: `let student = 'Max';`

## Loose Typing

- Wenn die Variable deklariert wird, wird kein Typ angegeben
- Erst bei der Definition entscheidet JavaScript automatisch anhand des Wertes, welchen Typ die Variable haben soll
- Der Variablen `student` aus dem oberen Beispiel hätte also auch der Wert `10` oder `true` zugewiesen werden können
- Beim Vermischen unterschiedlicher Datentypen in einer Variablen nimmt JavaScript automatisch Umwandlungen vor, was schnell zu Problemen führen kann

# JavaScript – Typen

- Variable haben immer einen bestimmten Datentyp (Number, String, Boolean, Array, Function, undefined…)
  - Dabei unterscheidet JavaScript nicht zwischen Integer, Long, Double usw. Alle Zahlen können mit oder ohne Dezimalstellen geschrieben werden

- Dynamische Typen
  - Hat eine Variable zunächst den Typ number (`let x = 5;`), kann ihr nachträglich noch ein anderer Wert eines anderen Typs zugewiesen werden (`x = 'Max';`)
  - In diesem Beispiel hat x zunächst den Typ `number`, anschließend `String`

- Automatische Typkonvertierung
  - Wenn Operatoren Variablen unterschiedlicher Typen übergeben werden, versuchen diese, die Typen der Variablen automatisch zu konvertieren
  - `let x = 5 + 'Max';` ergibt `"5Max"`
  - Auswertung von Ausdrücken dabei immer von links nach rechts
  - `let x = 16 + 4 + 'Max';` ergibt `"20Max"`
  - `let x = 'Max' + 16 + 4;` ergibt `"Max164"`

# JavaScript – Operatoren

- Arithmetische Operatoren und Zuweisungen

| Operator | Beschreibung |
|---|---|
| `+` | Addition |
| `-` | Subtraktion |
| `*` | Multiplikation |
| `/` | Division |
| `%` | Modulo |
| `++` | Inkrementieren |
| `--` | Dekrementieren |
| `x += y` | x = x + y |
| `x -= y` | x = x – y |
| `x *= y` | x = x * y |
| `x /= y` | x = x / y |
| `x %= y` | x = x % y |

# JavaScript – Operatoren

- Vergleichsoperatoren und logische Operatoren

| Operator | Beschreibung |
|---|---|
| == | Werte sind gleich |
| === | Werte und Typen sind gleich |
| != | Werte sind ungleich |
| !== | Werte ungleich oder Typen ungleich |
| > | Größer als |
| < | Kleiner als |
| >= | Größer oder gleich |
| <= | Kleiner oder gleich |

```
5 == '5';  ist true
5 === '5'; ist false
5 !== '5'; ist true
```

# JavaScript – Schleifen und Verzweigungen

## If – Bedingte Ausführung von Code

- Führe einen Anweisungsblock nur unter einer bestimmten Bedingung aus
- Der else-Block ist optional

```
if (Bedingung) {
    //Anweisungen
}  else {
    //Anweisungen
}
```

## While – Bedingte Schleife

- Führe einen Anweisungsblock aus, solange eine bestimmte Bedingung erfüllt ist

```
while (Bedingung) {
    //Anweisungen
}
```

## For – Zählschleife

- Führe einen Anweisungsblock aus, solange die Zählvariable einen bestimmten Wert nicht überschritten/unterschritten hat

```
for (let i = 0; i < 10;  i++) {
    //Anweisungen
}
```

# JavaScript – Funktionen

- Wiederverwendung von Anweisungsblöcken
  - Funktionen fassen Anweisungen zusammen und lassen sich dank Parametern vielseitig einsetzen
  - Schlüsselwort `function`

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

  - Die Funktion kann dann an beliebigen Stellen aufgerufen werden :

```
document.getElementById('demo').innerHTML =
                        'Das Ergebnis ist: ' + myFunction(4, 3);
```

# JavaScript – Funktionen

- JavaScript kennt vordefinierte Funktionen

| Funktion | Beschreibung |
|---|---|
| `decodeURI()` | Dekodiert eine URI |
| `encodeURI()` | Kodiert eine URI |
| `eval()` | Führt einen String aus, als wäre er JavaScript Code |
| `isFinite()` | Bestimmt, ob ein Wert eine endliche Zahl ist |
| `isNaN()` | Bestimmt, ob ein Wert keine Zahl ist |
| `Number()` | Konvertiert ein Objekt in eine Zahl |
| `String()` | Konvertiert ein Objekt in einen String |
| `parseFloat()` | Wandelt einen String in eine Gleitkommazahl um |
| `parseInt()` | Wandelt einen String in einen Integer um |

# JavaScript – Beispiel

Datei: beispiel.html

```
<!DOCTYPE html> <html>
  <head>
    <script src="beispiel.js"> </script>
  </head>
  <body>
    <button type="button" onclick="maxValue([1,5,-2,4])">
     Klick mich</button>
    <p id="demo"></p>
  </body>
</html>
```

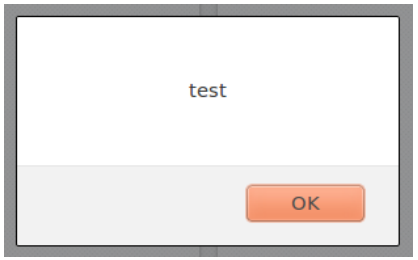

Datei: beispiel.js

```
function maxValue(pArray){
  var maxVal = pArray[0];
  for (var i = 1; i < pArray.length; i++) {
    if (maxVal < pArray[i]) {
      maxVal = pArray[i];
    }
  }
  document.getElementById('demo').innerHTML = 'Max Wert: ' + maxVal;
}
```

# JavaScript – Geltungsbereiche

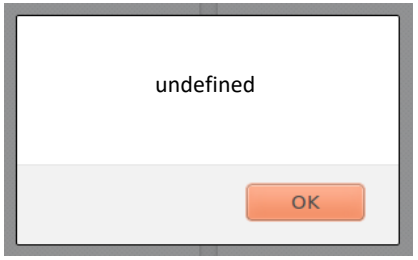
- Globaler Geltungsbereich
  - Variable, die außerhalb von Funktionen deklariert werden, sind im globalen Geltungsbereich
  - Auf den globalen Geltungsbereich kann von überall aus zugegriffen werden
- Lokaler Geltungsbereich
  - Funktionsparameter und Variable, die in Funktionen deklariert werden, sind nur innerhalb der Funktion sichtbar
  - Achtung: Wird innerhalb einer Funktion eine Variable ohne das Schlüsselwort `var` deklariert, so ist sie automatisch im globalen Geltungsbereich
- Funktions-Geltungsbereich(Hoisting) mit `var`

```
function test() {
    if(true) {
        var a = 'test';
    }
    alert(a);
}
```




- Block-Geltungsbereich mit `let`

```
function test() {
    if(true) {
        let a = 'test';
    }
    alert(a);
}
```

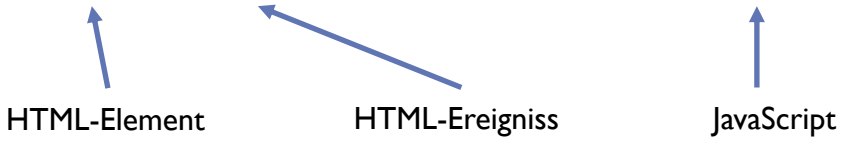


# JavaScript – Ereignisse

- HTML-Ereignisse
  - Immer wenn ein HTML-Ereignis ausgelöst wird, kann JavaScript ausgeführt werden
  - Es können Funktionen aufgerufen oder direkt JavaScript-Code ausgeführt werden
  - Ereignisse können aufgerufen werden, wenn eine HTML-Seite geladen wurde, ein Textfeld verändert wurde, ein Button geklickt wurde, …

- Syntax:
  - `<HTML-Element HTML-Event="JavaScript Code">`

```
<button onclick="getElementById('demo').innerHTML=Date()">Wie spät ist es?</button>
```



# JavaScript – Ereignisse

- Liste einiger wichtiger HTML-Ereignisse

| Ereignis | Beschreibung |
|---|---|
| onchange | Ein HTML-Element wurde verändert |
| onclick | Der Benutzer hat ein HTML-Element geklickt |
| onmouseover | Der Benutzer hat die Maus über ein HTML-Element bewegt |
| onmouseout | Der Benutzer hat die Maus von einem HTML-Element runter bewegt |
| onkeydown | Der Benutzer hat eine Taste auf der Tastatur gedrückt |
| onload | Der Browser hat die Seite fertig geladen |

# JavaScript – Objekte

- JavaScript-Objekte enthalten Schlüssel-Wert-Paare
  - Sie können Variablenwerte (Attribute) oder Funktionen (Methoden) enthalten
  - Neues Objekt anlegen (Objekt-Literal)

```
var student1 = {};
```

  - Attribute hinzufügen

```
student1.vorname = 'Max';
student1.nachname = 'Muster';
```

  - Methoden hinzufügen

```
student1.begruessung = function (){
    alert('Hallo, ich bin ' + this.vorname);
};
```

  - **`this`** bezieht sich immer auf das aufrufende Objekt
  - Aufruf der Methode mit `student1.begruessung();`

# JavaScript – Objekte: Assoziative Arrays / Maps

- JavaScript-Objekte sind assoziative Arrays
  - Zugriff auf Attribute mit [] möglich:

```
var student1 = {
    vorname: 'Susi',
    nachname: 'Sommer',
};

Var nn = student1.nachname;

var vn = student1['vorname'];

student1['matnr'] = '12345';
```

  - Jedes JavaScript-Objekt ist eine Map. Attributnamen sind die Schlüssel, Attributwerte sind die Werte. Der Zugriff ist in konstanter Zeit möglich.

# JavaScript – Objekte: Konstruktoren

- Konstruktoren (Konstruktor-Funktion)
  - Objektkonstruktoren vereinfachen das Erzeugen vieler Objekte der gleichen Art

```
function student(pVorname, pNachname, pMatrikelnummer){
    this.vorname = pVorname;
    this.nachname = pNachname;
    this.matrikel = pMatrikelnummer;
    this.begruessung = function (){
        alert('Hallo, ich bin ' + this.vorname);
    }
}
```

  - Nun lassen sich beliebig viele Studentenobjekte auf folgende Weise erstellen

```
var studentMax = new student('Max', 'Muster', 123456);
var studentPeter = new student('Peter', 'Soundso', 654321);
```

# JavaScript – new im Konstruktor erzwingen

- Vermeiden, dass new vergessen wird:

```
function student(pVorname, pNachname, pMatrikelnummer) {
    if (!(this instanceof student)) {
        return new student(pVorname, pNachname, pMatrikelnummer);
    }
    this.vorname = pVorname;
    this.nachname = pNachname;
    this.matrikel = pMatrikelnummer;
    this.begruessung = function (){
        alert('Hallo, ich bin ' + this.vorname);
    }
}
```

- Nun liefert der Aufruf des Konstruktors immer ein student-Objekt

```
var studentMax = new student('Max', 'Muster', 123456);
var studentPeter = student('Peter', 'Soundso', 654321);
```

# JavaScript – Typen und Typkonvertierung

- Datentypen
  - JavaScript kennt 5 Datentypen, die Werte enthalten: `string, number, boolean, object, function`
  - Datentypen, die keine Werte enthalten können: null, undefined
- Der typeof-Operator
  - Ermittelt den Datentyp einer Variable und gibt diesen als String zurück
  - `typeof 'Max' //String`
  - `typeof 3.14 //number`
  - `typeof {name: 'Max', alter: 23} //object`
  - `typeof [1,2,3] //object`
  - `typeof new Date() //object`
  - `typeof function () {} //function`
  - `typeof student //undefined (falls noch nicht deklariert)`
  - `typeof null //object`

# JavaScript – Typen und Typkonvertierung

- Manuelle Typkonvertierung
  - Mithilfe einer Vielzahl von JavaScript-Funktionen lassen sich Datentypen manuell konvertieren
  - number zu string: `String(123)` oder `(123).toString //"123"`
  - boolean zu string: `String(true)` oder `(true).toString //"true"`
  - string zu number: `Number("3.14") //3.14`
  - boolean zu number: `Number(false) //0 bzw. 1 bei true`
- Automatische Typkonvertierung
  - JavaScript versucht Datentypen, die nicht zusammenpassen, automatisch zu konvertieren
  - Die Ergebnisse sind dann nicht immer wie erwartet
  - `5 + null //5, da null zu 0 konvertiert wird`
  - `'5' + null //"5null", da null zu "null" konvertiert wird`
  - `'5' + 1 //"51", da 1 zu "1" konvertiert wird`
  - `'5' - 1 //4, da "5" zu 5 konvertiert wird`

# JavaScript – Debugging

- Jeder Programmierer macht Fehler
  - In JavaScript sind diese ohne Hilfsmittel oft schwer zu finden
  - Wenn JavaScript-Code Fehler enthält, gibt es oft keine Fehlermeldungen oder Hinweise darauf, wo sich der Fehler befinden könnte

- Hilfsmittel für die Fehlersuche
  - Alle modernen Browser verfügen über integrierte Debugger, welche manuell aktiviert werden müssen
  - Der Debugger ermöglicht es, die Codeausführung an einer oder mehreren Stellen zu unterbrechen, um z.B. Variablen zu untersuchen
  - Chrome: Menü → Weitere Tools → Entwicklertools

# JavaScript – Debugging

- console.log()
  - Die Konsole befindet sich in den Entwicklertools
  - Die JavaScript-Methode console.log() kann Text in diese Konsole schreiben
  - So lassen sich wichtige Stellen im Programm überwachen
- Breakpoints
  - Stellen, an denen die Ausführung des Codes zum Debuggen unterbrochen wird, heißen Breakpoints

```
<script>
    var a = 5;
    …
    console.log(a);
</script>
```

# JavaScript – Codequalität

- JavaScript verführt oft dazu, Code von schlechter Qualität zu erstellen
- strict mode
  - Durch die "use strict"-Direktive lässt sich der strict mode in JavaScript aktivieren
  - Die Direktive aktiviert den strict mode für den jeweiligen Geltungsbereich
  - Im strict mode gibt JavaScript Fehlermeldungen u.a. in folgenden Fällen in die Konsole aus
    - Benutzung undeklarierter Variablen
    - Mehrfachdefinition von Attributen
    - Mehrfachbenutzung des selben Parameternamens

```
<script>
    "use strict";
    x = 3.14;
    var y = {a1: 'a', a1: 'b'};
    function f(p1, p1) {};
</script>
```

# JavaScript – Codequalität

- Konventionen und Programmierstil
  - Variablen- und Funktionsnamen sinnvoll vergeben
  - camelCase für Variablen- und Funktionsnamen benutzen (z.B. pFirstName)
  - Leerzeichen vor und hinter Operatoren ( `x + (y * z)` statt `x+(y*z)` )
  - Lange Parameterlisten bei Funktionen vermeiden
  - Globale Variable vermeiden
  - Variable am Anfang deklarieren und initialisieren
  - JavaScript-Code in externe Dateien auslagern
  - Korrekte Dateiendungen benutzen (`.html, .css, .js, …`)
- Häufige Fehler
  - Unerwartete Typkonvertierung
  - Falsche Benutzung von "=" oder "===" in if-Bedingungen
  - Verwechslung von Addition und Konkatenation ("+"-Operator)
  - Verwechslung von `undefined` und `null`
  - Falsch interpretierte Geltungsbereiche

# JavaScript – Unerwartetes Verhalten

- http://blog.mgechev.com/2013/02/22/javascript-the-weird-parts/

```
1 == 1; //true
'foo' == 'foo'; //true
[1,2,3] == [1,2,3]; //false
```

```
new Array(3) == ",,"; //true
```

```
1 * 'foo'; //NaN
1 * 2; //2
'foo' * 1; //NaN
'foo' + 1; //'foo1'
```

```
{} + {} //NaN
```

```
[] + []; //""
```

```
[] + {}; //[object Object]
```

# Webapplikationen mit JavaScript

DOM – Document Object Model

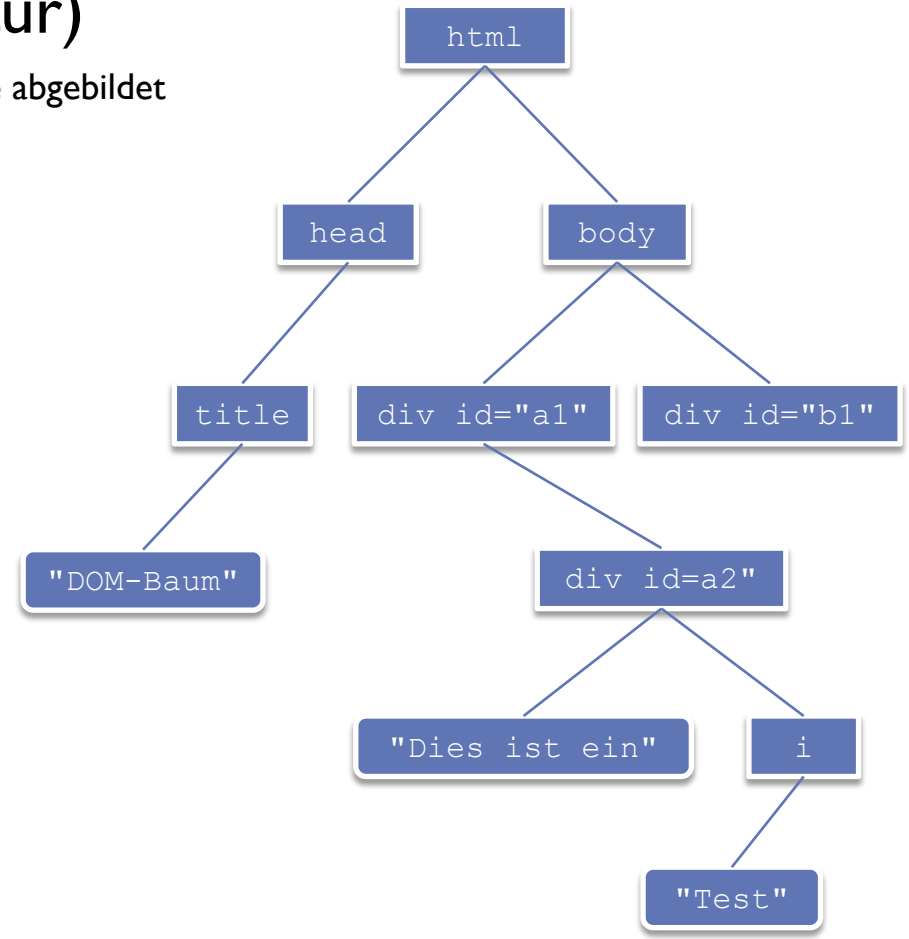
# DOM – Einleitung

- DOM ist ein W3C-Standard
- Bildet die einzelnen Elemente einer Webseite in einer Objekthierarchie ab
- Zuvor hatten alle Browserhersteller ihre eigenen Techniken, weshalb es große Unterschiede zwischen den Browsern gab
- Erst durch den DOM-Standard wurde es möglich, browserunabhängige Skripte zu programmieren
  - Gängige Browser unterstützen weitestgehend DOM Level 2
  - DOM Level 3 funktioniert noch nicht komplett in den gängigen Browsern

# DOM – Aufbau eines Dokuments

- Objekte einer Webseite werden in einer Objekthierarchie dargestellt (Baumstruktur)
  - Sämtliche Elemente werden als Objekte abgebildet

```
<html>
  <head>
    <title>DOM-Baum</title>
  </head>
  <body>
     <div id="a1">
        <div id="a2">
           Dies ist ein <i>Test</i>
        </div>
     </div>
     <div id="b1">
     </div>
  </body>
</html>
```



# DOM – Elemente ansprechen

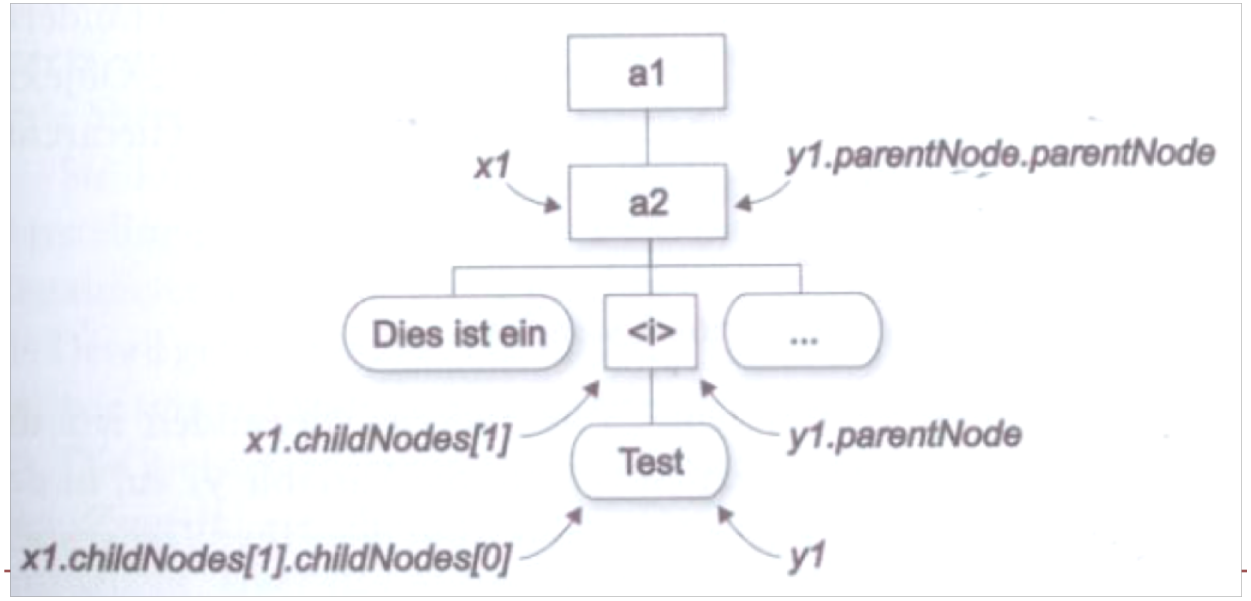
- Elemente direkt ansprechen
  - **`getElementById()`**
    - erwartet als Argument einen String mit dem ID-Attribut und liefert Referenz auf das gesuchte Objekt
    - `var x1 = document.getElementById('a2');`
  - **`getElementsByTagName()`**
    - gibt ein Array mit allen gefundenen Objekten des angegebenen Typs zurück
    - `var d = document.getElementsByTagName('div')`
    - Mit `d[0]` könnten wir dann auf das erste -Element zugreifen

# DOM – Elemente ansprechen

- Vordefinierte Arrays verwenden
  - Es gibt vordefinierte Arrays, mit denen auf einzelne Objekte zugegriffen werden kann
    - Auf das erste Bild einer Webseite greift man mit `document.images[0]` zu
    - Die Breite des Bildes erhält man mit `document.images[0].width`
  - Alle Links einer Webseite sind im `links`-Array enthalten
  - Das `forms`-Array enthält alle Formulare
  - Im `elements`-Array sind die einzelnen Formularelemente enthalten
  - Wenn eindeutige IDs verwendet wurden, kann man auch `document.images['bild1']` verwenden
    - wobei „`bild1`“ der ID des Elements entspricht
    - Vorteil dabei ist, dass die Reihenfolge der Elemente keine Rolle mehr spielen
    - Bei Änderungen der Reihenfolge muss nicht der ganze Code angepasst werden

# DOM – Elemente ansprechen

- Kinder und Eltern ansprechen
  - Das `childNodes`-Array enthält alle direkten Kinder des aktuellen Elements
  - Das erste Kind eines Elements erhält man mit `einKnoten.childNodes[0]`
  - Das erste Element im `childNodes`-Array lässt sich auch mit `einKnoten.firstChild` und das letzte mit `einKnoten.lastChild` ansprechen
  - Das Elternelement wird mit `einKnoten.parentNode` angesprochen



# DOM – Elemente verändern

- Inhalt eines Textelements ändern
  - `einTextElement.nodeValue = 'Versuch';`
- Inhalt eines HTML Elements ändern
  - Den enthaltenen HTML-Code eines Elements bekommt man mit `einElement.innerHTML`
  - Dieser kann verändert oder komplett neuer HTML-Code hinzugefügt werden
  - Einfacher Weg, HTML-Code an einer bestimmten Stelle einzufügen
  - z.B. `einElement.innerHTML = '<b>Hallo!</b>';`

# DOM – Elemente verändern

- Darstellung eines Elements ändern
  - Man kann auf Stylesheets zugreifen und sie verändern
  - Die meisten Elemente haben ein Style-Objekt, welches den Zugriff auf die Stylesheet-Definition eines Elements ermöglicht
  - Die Farbe eines Elements z.B. ändert man mit

    ```
    einElement.style.color='red';
    ```

  - Die Stylesheet-Klasse kann mit

    ```
    einElement.className='rot';
    ```

    festgelegt werden, wenn zuvor eine Stylesheet-Klasse `.rot{}` definiert wurde

# DOM – Elemente hinzufügen

- Textelement hinzufügen
  - Ein neues Textelement wird mit

    ```
    textElement=document.createTextNode('abc');
    ```

    erzeugt
  - Der Inhalt des Textelements ist `abc`
  - Noch ist der neue Text nicht sichtbar
  - Er muss an den DOM-Baum angehängt werden
  - Das anhängen an ein beliebiges Element geschieht mit

    ```
    einElement.appendChild(textElement);
    ```

# DOM – Elemente hinzufügen

- Andere Elemente hinzufügen
  - Alle anderen Elemente werden mit

    ```
    var einElement=document.createElement('elementName');
    ```

    erzeugt
  - Beispiel:

    ```
    var einNeuesElement=document.createElement('div');
    ```
  - Das Anhängen geschieht mit

    ```
    einElement.appendChild(einNeuesElement);
    ```

- Attribute hinzufügen
  - Mit der Methode setAttribute() kann einem Element ein Attribut zugewiesen werden
  - `einImageElement.setAttribute('src', 'bild.gif')`

    fügt dem Element das Attribut `src` mit dem Wert `bild.gif` hinzu

# DOM – Elemente umhängen

- Elemente umhängen
  - Die Methode `appendChild()` kann auch auf bereits im DOM-Baum eingebundene Elemente angewendet werden
  - `appendChild()` sorgt in diesem Fall dafür, dass ein Element umgehängt und an der alten Stelle gelöscht wird

```
var einElement = document.getElementById('b1');
var einAnderesElement = document.getElementById('b2');
einElement.appendChild(einAnderesElement);
```

  - `einAnderesElement` wurde an `einElement` gehängt und an der alten Stelle gelöscht

# DOM – Elemente kopieren

- Elemente kopieren
  - Soll ein Element kopiert werden, kann die Methode `cloneNode()` verwendet werden
  - Als Parameter wird ein boolescher Wert übergeben, der angibt, ob alle untergeordneten Elemente mit kopiert werden sollen

```
var einElement = document.getElementById('b1');
var einAnderesElement = document.getElementById('b2');
einElement.appendChild(einAnderesElement.cloneNode(true));
```

  - `einAnderesElement` wurde an `einElement` gehängt und bleibt an der alten Stelle bestehen

# DOM – Elemente entfernen

- removeChild() entfernt ein Element aus dem DOM-Baum
  - `einElement.removeChild(einAnderesElement);`
  - Der komplette Ast mit allen darunter liegenden Elementen wird entfernt
  - Sollen die untergeordneten Elemente nicht entfernt werden, müssen sie vorher umgehängt werden

# JSON

JavaScript Object Notation

# JSON

- Was ist JSON?
  - JSON: **J**ava**S**cript **O**bject **N**otation
  - Eine Syntax zum Speichern und Austauschen von Daten
  - Eine einfachere Alternative zu XML
  - Sehr stark an JavaScript-Syntax angelehnt, dennoch programmiersprachenunabhängig
  - Lässt sich in AJAX verwenden (anstatt XML)

- Warum JSON?
  - JSON-Notation ist kürzer und einfacher zu lesen und zu schreiben
  - Kann mit Arrays umgehen
  - Kann von JavaScript Standardfunktionen ausgelesen werden
  - Intuitiv und selbsterklärend
  - Bei AJAX-Anwendungen schneller und einfacher als XML

# JSON Syntax

- Syntaxregeln
  - Objekte sind Schlüssel/Werte-Paare
  - Attribute sind kommasepariert
  - Geschweifte Klammern repräsentieren Objekte
  - Eckige Klammern repräsentieren Arrays

- Syntaxbeispiel
  - `{"firstName":"John", "lastName":"Doe" }`
  - Definiert ein JSON-Objekt mit den Attributen "firstName" und "lastName"
  - Mit dieser Syntax lassen sich nicht nur JSON-Objekte, sondern auch JavaScript-Objekte erzeugen

# JSON Syntax

- JSON Arrays
  - Arrays stehen in eckigen Klammern und können mehrere Objekte enthalten
  - In JavaScript kann JSON-Notation genutzt werden, um ein Array zu erstellen
  - Folgendes ist valider JavaScript-Code. Alles was rechts des Gleichzeichens steht, ist auch korrekte JSON-Syntax

```
var employees = [ { "firstName" : "John" , "lastName" : "Doe"},
                  { "firstName" : "Anna" , "lastName" : "Smith"},
                  { "firstName" : "Peter" , "lastName" : "Jones"} ];
```

# JSON vs XML

## XML-Notation

```
<employees>
    <employee>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName>
        <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName>
        <lastName>Jones</lastName>
    </employee>
</employees>
```

## JSON-Notation

```
{"employees":[
    {"firstName":"John",
     "lastName":"Doe"},
    {"firstName":"Anna",
     "lastName":"Smith"},
    {"firstName":"Peter",
     "lastName":"Jones"}
]}
```

- Beide Beispiele sind semantisch identisch

# JSON Zugriff auf Objekte

```
var person = {
          name: 'John Johnson',
          street: 'Oslo West 16',
          phone: '555 1234567'
};

var jsonPerson = JSON.stringify(person);

// jsonPerson enthält den String
// '{"name":"John Johnson","street":"Oslo West a16","phone":"555 1234567"}'

var person2 = JSON.parse(jsonPerson);
```