JavaScript in Stichworten

- Interpretierte Sprache
- Ermöglicht prozedurale, objektorientierte und funktionale Programmierung
- Anfangs nur zur Webclient-Programmierung (Dynamisierung von Webseiten)
  inzwischen auch Standalone und auf dem Server verfügbar
- 1985 bei Firma Netscape von Brendan Eich erfunden
- Standardisierte Versionen heißen ECMAscript

Weblinks

- JavaScript-Abteilung bei selfhtml.org
- JavaScript-Abteilung bei developer.mozilla.org
- JavaScript bei W3schools (englisch)
- Interaktiver Programmierkurs:
  Google Grasshopper (englisch)

Formatierung

JavaScript ist frei formatierbar.
Konventionen sollten beachtet werden.

- Nur ein Statement pro Zeile
- Blockstruktur mit Einrückungen deutlich machen
- Weitere Konventionen des Programmierprojektes beachten

Ein guter Texteditor unterstützt beim Einhalten der Konventionen.

Werte

Zahlen

```
100000.9999
```

- Dezimaltrenner ist der Punkt, Tausendertrennzeichen werden nicht verwendet.
- JavaScript rechnet immer mit Fließkommazahlen.

Texte (Strings)

```
'<link a href="...">'
```

- Text wird mit Hochkommas oder Anführungszeichen begrenzt.
- Beim Erzeugen von HTML sind Hochkommas geschickter.
- In JSON muss man Anführungszeichen verwenden.

Kommentare

- Zwei Schrägstriche `//` leiten einen Kommentar ein, der bis zum Zeilenende reicht.
- `/*` leitet einen Kommentar ein, der bis zu `*/` reicht. Er kann sich über mehrere Zeilen erstrecken.

Variablen

- Variablen in JavaScript haben einen begrenzten Gültigkeitsbereich.
- Variablen in JavaScript sind dynamisch typisiert.
  Erst zur Laufzeit wird überprüft, ob die gewünschte Operation möglich ist.
- Deklarieren Sie Variablen vor oder beim ersten Gebrauch mit `var` oder `let`.
- Schreiben Sie am Anfang Ihre Skriptes `'use strict';`, um Deklarationsfehler abzufangen.

```
var nachname;              // nur deklariert
var vorname = 'Fred';      // gleichzeitig zugewiesen
```

Variablennamen

- Variablennamen beginnen mit einem (kleinen) Buchstaben. Sie können auch Ziffern und Unterstriche enthalten.
- Groß- und Kleinschreibung wird unterschieden.
- Wählen Sie sinnvolle und eindeutige Variablennamen.
- Lange Namen werden mit Unterstrichen oder CamelCase lesbar gemacht: `steuersatz_voll_prozent` oder `steuersatzVollProzent`

Zuweisungen

Zuweisungen unterscheiden sich von Gleichungen:

- Links des = muss eine Variable stehen (ein *lvalue*).
- Die Variable erhält den momentanen Wert des rechten Ausdrucks.
  Spätere Änderungen des rechten Ausdrucks wirken sich nicht aus.
- Einer Variablen können mehrmals Werte zugewiesen werden.
  Sie gelten bis zur nächsten Zuweisung.
- Wenn Sie einen undefinierten Wert zuweisen, erhalten Sie eine Fehlermeldung.

```
x = 3.5;
```

Rechenzeichen

- `+ - * /` und `( )` funktionieren wie erwartet.
- Dazurechnen:
  `a += b;` kürzer für `a = a + b;`
  Entsprechendes mit anderen Operatoren: `-= *= /=`
- Inkrementieren (um 1 erhöhen):
  `a++;` kürzer für `a = a + 1;` (oder `a += 1;`
- Dekrementieren: `a--`

Objekte

- Objekte sind Variablen, die mehrere zusammengehörige Werte (Eigenschaften) enthalten.
- Eigenschaften tragen jeweils einen Namen.
- Eigenschaften werden mit `.eigenschaft` oder `['eigenschaft']` angesprochen.
- In vielen Programmiersprachen basieren Objekte auf Klassen, in JavaScript nicht.

Objekt

Definition

```
var auto = {
  marke: 'VW',
  modell: 'Golf',
  ps: 120 };
```

Zugriff

```
console.log(auto.marke);
console.log(auto['marke']);
auto.marke = 'Audi';
auto['verbrauch'] = 10;
```

Arrays

- Ein Array ist eine Liste von Werten.
- Wie bei einer Reihe geparkter Autos ist es einfach, am Anfang oder Ende Werte zuzufügen oder zu entfernen, in der Mitte ist es komplizierter.
- Arrays werden als Aufzählungen innerhalb eckiger Klammern formuliert.
- Arrays können weitere Arrays enthalten.

```
var noten = [3, 1.5, 2];  // einfaches Array
```

Array-Elemente

```
var dichter = ['Rilke', 1875, 1926];
```

- Arrayelemente werden beginnend mit 0 durchgezählt.
- Auf Arrayelemente wird mit eckigen Klammern `[ ]` zugegriffen.
- Die Länge eines Arrays erhält man mit `.length`
- Die Nummern der Array-Elemente laufen von 0 bis .length-1

Inhalt	'Rilke'	1875	1926
Index	0	1	2

```
let dichterName = dichter[0]
let anzahlDaten = dichter.length; // ergibt 3
```

Arrays verändern (1)

```
var dichter = ['Rilke', 1875, 1926];
```

- Am Ende anhängen: `dichter.push('Stunden-Buch');`
- Vom Ende entfernen: `werk = dichter.pop();`
- Am Anfang einfügen: `dichter.unshift('Rainer');`
- Vom Anfang entfernen: `vorname = dichter.shift();`
- Einen Wert ersetzen: `dichter[0] = 'Goethe';`

Arrays verändern (2)

- Werte ausschneiden und einfügen:
  `splice( abwo, wieviele , Ersatz ...);`
  - *abwo*: Index des ersten zu entfernenden Elements
  - *wieviele*: Anzahl zu entfernender Elemente
  - *Ersatz*: einzufügendes Element

Arrays verändern (3)

map/reduce

```
kostenInCent = [200, 300, 50];
kostenInEuro = kostenInCent.map(x => x / 100);
```

- `map()` führt eine Berechnung für jedes Element in einem Array aus.
- Es wird ein neues Array zurückgegeben.
- Das Argument von `map()` ist eine Pfeilfunktion.

Arrays auswerten

```
kostenInCent = [200, 300, 50, 400];
```

- Einen Wert finden:
  ```
  stelle = kostenInCent.indexof(50); // ergibt 2
  ```
- Einen Ausschnitt kopieren:
  ```
  jahre = kostenInCent.slice(1, 3);
  // ergibt [300, 50]
  ```
- Liste filtern:
  ```
  kleinBetraege = kostenInCent.filter(
    betrag => betrag < 100); // ergibt [50]
  ```

Strings

```
var titel='<h2 class="wichtig">Tipps</h2>';
var anrede = "Herr";
```

- String ist der Fachausdruck für Texte.
- String-Behandlung ähnelt Array-Behandlung.
  (Man kann einen String als Array von Buchstaben sehen.)
- Auch einfache Strings können in JavaScript wie Objekte verwendet werden.

```
var laenge = anrede.length; // 4
var laenge2 = 'Damen und Herren'.length; // 16
```

Strings untersuchen

```
var anreden = "Herr, Frau, Damen und Herren";
```

- Anzahl Zeichen: `.length`
- Wo enthält er einen anderen String?
  ```
  var erstermann = anreden.indexOf("Herr"); //
  ergibt 0
  var letztermann =
  anreden..lastIndexOf("Herr"); // ergibt 22
  ```
- Wo enthält er ein Muster?
  `.search()`

Strings teilen

```
var anreden = "Herr, Frau, Damen und Herren";
```

- Ein Stück von n bis m kopieren: `.slice(n, m)`

```
var anrede2 = anreden.slice(6,10);
// ergibt "Frau"
```

- Den Rest kopieren:

```
var anrede3 = anreden.slice(12);
// ergibt "Damen und Herren"
```

Strings ändern

```
var anreden = "Herr, Frau, Damen und Herren";
```

- Suchen und Ersetzen:
  ```
  var anrede2 = anreden.replace('Herr', 'Mann');
  ```
  Nur das erste Vorkommen wird ersetzt.
- Suchen und Ersetzen:
  ```
  var anrede2 = anreden.replaceAll('Herr',
  'Mann');
  ```
  Jedes Vorkommen wird ersetzt.
- In Großbuchstaben wandeln:
  ```
  var laut = 'Mann'.toUpperCase(); // 'MANN'
  ```
- In Kleinbuchstaben wandeln:
  ```
  var laut = 'Mann'.toLowerCase(); // 'mann'
  ```

Strings stutzen und auffüllen

- Leerzeichen an Anfang und Ende entfernen: `(.trim()`
- Mit Nullen auffüllen (von vorn)
  `neu = '17'.padStart(4, '0'); // ergibt 0017`
- Entsprechend von hinten mit `padEnd()`

Funktionen

```
zeichneLinie(0.0, 0.0, 3.0, 2.0);
```

- Funktionsnamen beginnen mit einem Buchstaben.
- Funktionsargumente werden in runden Klammern aufgeführt und mit Kommas getrennt.
- Wenn es sinnvoll ist, sollte eine Funktion etwas zurückliefern.
- Der Rückgabewert einer Funktion muss nicht ausgewertet werden.
- Wählen Sie aussagekräftige Funktionsnamen.
- Funktionen, die auf eine Bildschirmseite passen, sind besser als größere.

Funktion definieren

- Das Schlüsselwort `function`, dann der (selbst gewählte) Funktionsname, dann runde Klammern mit Argumentnamen.
- Funktionsinhalt in geschweifte Klammern.
- Variablen am Funktionsanfang mit `let` oder `var` definieren.
- Rücksprung aus der Funktion mit `return`.

```
function multipliziere(x, y)
{
    let ergebnis = x * y;
    return ergebnis;
}
```

Rekursion

- Eine rekursive Funktion ruft sich selbst auf.
- Klare Endbedingungen müssen existieren, damit die Aufrufe irgendwann aufhören.
- Rekursion sieht elegant aus, ist aber meist nicht effizienter als andere Verfahren.

Rekursion: Beispiel

Alle Werte aus einem Array, das auch Arrays enthält, sollen ausgegeben werden.

```
var daten = ['a1', 'a2', ['b3', 'b4', 'b5'],
  'a6', ['b7', ['c8', ['d9', 'd10']]], 'a11'];
function druckeArray(liste) {
  for(let element of liste) {
    if(Array.isArray(element)) {
      druckeArray(element);
    } else {
      console.log(element);
    }
  }
}
druckeArray(daten);
```

Arrays durchgehen (1)

```
var noten = [1.5, 2.5, 2.0];
```

Moderne for-Schleife

```
var summe = 0, durchschnitt;
for(var element of noten) {
    summe += element;
}
durchschnitt = summe / noten.length;
```

Arrays durchgehen (2)

```
var noten = [1.5, 2.5, 2.0];
```

Klassische for-Schleife

```
var summe = 0, durchschnitt;
for(let i = 0; i < noten.length; i++) {
    summe += noten[i];
}
durchschnitt = summe / i;
```

Arrays durchgehen (3)

```
var noten = [1.5, 2.5, 2.0];
```

Map/Reduce ist komplizierter als eine klassische Schleife, aber es kann parallel verarbeitet werden und ist deswegen vielleicht schneller.

Map/Reduce

```
var summe = 0, durchschnitt;
const summiere = (summe, wert) => summe + wert;
gesamt = noten.reduce(summiere, 0);
durchschnitt = gesamt / noten.length;
console.log(durchschnitt);
```

Pfeil-Funktionen

```
const summiere = (summe, wert) => summe + wert;
gesamt = noten.reduce(summiere, 0);
```

- An `reduce()` muss eine Funktion übergeben werden.
- Anzahl und Reihenfolge der Parameter sind vorgegeben.
- `summiere` ist eine Variable, die eine anonyme Funktion enthält.
- Beachten Sie das `=`, den fehlenden Funktionsnamen zwischen `summiere` und runder Klammer und das `=>` statt geschweifter Klammern um den Funktionsrumpf.
- Pfeil-Funktionen heißen auch Lambda-Ausdrücke oder Closures.

Vergleiche

>	größer
>=	größer oder gleich
<	kleiner
<=	kleiner oder gleich
!=	Wert ungleich
!==	Wert oder Typ ungleich
==	Wert gleich
===	Wert und Typ gleich

Wahre Werte

- Gibt man in einer Entscheidung nur einen Wert statt eines Ausdrucks an, wird dieser Wert bewertet.
- 0, NaN[1], leerer Text und undefiniert gilt als FALSCH (ist falsy).
- Alles andere gilt als WAHR (ist truthy).

```
var nummer = -1;
if(nummer) { ... }   // ergibt wahr
```

[1] Not a Number, wird von mathematischen Funktionen geliefert, wenn das Ergebnis ungültig ist.

Entscheidungen

- Entscheidungen mit Blockstruktur: `if(...) { ... }` oder `if(...) { ... } else { ... }`
- Inline-Entscheidungen mit `? :`

```
eintritt = alter >= 18 ? 'voll' : 'erm.';
```

JavaScript einbinden

1. JavaScript im `<head>` aus einer externen Datei laden:
   `<script src="myScript.js"></script>`
2. JavaScript als Block direkt einbinden:
   `<script> /* JavaScript-Code */ </script>`
3. Script-Elemente direkt in Tags anbringen:
   `<button onclick="meineFunktion()">Hier klicken</button>`

Möglichkeit 3 sollte vermieden werden.

Probleme

- JavaScript wird ausgeführt, sobald es beim Browser ankommt (auch, wenn die Seite noch nicht komplett geladen ist).
- JavaScript und HTML sollten nicht bunt gemischt werden. Der Code steht bestenfalls an *einer* Stelle oder in einem externen Skript.

Vorgehen

- Der meiste JavaScript-Code steckt in Funktionen.
- Der einzige JavaScript-Code, der direkt ausgeführt wird, installiert einen Ereignis-Handler, der beim vollständigen Laden des Dokuments getriggert wird.
- Dieser Handler bindet weiteren JavaScript-Code an Dokumenten-Elemente und Ereignisse.

Vorgehen: Beispiel

```
<script>
document.addEventListener('DOMContentLoaded',
    function () {
  document.querySelector('#interaktiv').
    addEventListener('click', handleKlick);
  function handleKlick() {
    alert("Hallo Welt!");
  }
});
</script>
<button id="interaktiv"type="button">
```

Vorgehen: Beispiel

1. anonyme Funktion definieren und als Ereignishandler registrieren:
```
document.addEventListener('DOMContentLoaded',
  function () {
```
2. Diese Funktion registriert einen Handler, der beim Klick auf das Element mit `id="interaktiv"` die Funktion `handleKlick()` aufruft:
```
  document.querySelector('#interaktiv').
  addEventListener('click', handleKlick);
```
3. Definition der Funktion `handleKlick()`:
```
function handleKlick() {
  alert("Hallo Welt!");
}
```
4. Klammern schließen:
```
});
```

Events auswerten

- Ereignishandler bekommen ein *Event* als Argument. Damit kann man erfahren, welches Element das Ereignis ausgelöst hat (`event.target`).
- Je nach Event kann man mehr erfahren – bei Maus-Events zum Beispiel, wo genau geklickt wurde.

```
function liesWert(event) {
  let wert = event.target.value;
  ...
}
```

Typische Events

blur	Verlassen eines Feldes
change	Ein Feld wurde geändert
click	Es wurde geklickt
DOMContentLoaded	Dokument ist komplett geladen
focus	Ein Feld wird aktiviert
input	Ein Feld wird bearbeitet
mousemove	Maus wird bewegt
submit	Absenden-Knopf wurde geklickt

Zugriff auf Seitenelemente

- Der Browser stellt die Seite als Baumstruktur zur Verfügung.
- Auf Elemente kann über ihre ID, den Namen, die Klasse, den HTML-Tag oder über weitere CSS-Selektoren zugegriffen werden.
- Je nach Methode erhält man ein eindeutiges Ergebnis, das erstebeste Ergebnis oder ein Array von Elementen..

Zugriff auf Seitenelemente

HTML-Code:

```
<input type="text" id="vnFeld" name="vorname">
<input type="text" id="nnFeld" class="nn"
  name="nachname">
```

JavaScript-Zugriff auf das Nachnamenfeld:

```
let f1 = document.getElementById('nnFeld');
let f2 = document.getElementsByName('nachname')[0];
let f3 = document.querySelector('input.nn');
let f4 = document.querySelectorAll('input.nn')[0];
```

Zugriff auf Seitenelemente

Unterschiedliche Methoden liefern unterschiedliche Ergebnisse:

Methode	*Ergebnis*	*Element*
getElementById()	Element	direkt
getElementsByName()	NodeList	Index []
querySelector()	Element[2]	direkt
querySelectorAll()	NodeList	Index []

[2] erstes passendes Element

Zugriff auf Attribute

```
<input type="text" id="vnFeld" name="vorname">
<input type="text" id="nnFeld" class="nn"
  name="nachname" value="Maier"() >
```

Attribute können wie Objekteigenschaften gesetzt und gelesen werden:

```
let f1 = document.getElementById('nnFeld');
console.log('Er heißt ' + f1.value);
f1.value = 'Fischer';
// Ab jetzt heißt er Fischer.
```

Stil ändern

- CSS-Eigenschaften sind über .style erreichbar.
- Statt CSS-Bindestrichen wird interner CamelCase verwendet.
- Der Stil kann über die Eigenschaft `cssText` auch direkt mit CSS gesetzt werden.

```
let testfeld = document.getElementById('test');
testfeld.style.backgroundColor = 'red';
// alternativ mit []-Klammern
testfeld.style['background-color'] = 'red';
// oder mit Standard-CSS-Ausdrücken:
testfeld.style.cssText = 'background-color: red;';
```

Zugriff auf Inhalte

- Manipulation von innerHTML oder innerText.
- Blätter im Baum einfügen mit appendChild();
- Knoten einfügen mit insertBefore();

appendChild()

```
const p = document.createElement('p');
document.body.appendChild(p);
```