

Prof. Dr.-Ing. Georg J. Schneider

Multimedia und Medieninformatik
Fachbereich Informatik
Hochschule Trier

JavaScript Literatur



JavaScript Programmieren für Einsteiger:
Der leichte Weg zum JavaScript-Experten
Paul Fuchs



JavaScript

http://openbook.rheinwerk-verlag.de/javascript_ajax/

Selfhtml - JavaScript

<https://wiki.selfhtml.org/wiki/JavaScript>

European Computer Manufacturers Association
<http://www.ecma-international.org/>

JavaScript

Was ist JavaScript?

- Von der Firma Netscape entwickelt
- Meistverwendete Skriptsprache im Internet
- Wie wird (X)HTML direkt vom Browser interpretiert

Was Sie lernen

- Was ist JavaScript?
- Erstes JavaScript-Programm
- JavaScript-Sprachelemente
- Document Object Model
- Event-Handler in JavaScript
- Formulare mit JavaScript validieren

Was ist JavaScript?

JavaScript-Versionen

- Standardisierungsgremium ECMA (European Computer Manufacturers Association)
- Aktuellste Version ist 11: ECMAScript 2020
- Vergleich bzgl. Browserunterstützung:
<http://kangax.github.io/compat-table/es2016plus/>

JavaScript einsetzen

- Website auch ohne JavaScript nutzbar!
- JavaScript wird von einigen Benutzern deaktiviert
- Websites ggf. so gestalten, dass Sie auch bei abgeschaltetem JavaScript noch funktionieren
- Verordnung zur Barrierefreiheit (BITV) fordert, dass eine Website ohne JavaScript zwar weniger attraktiv sein kann, aber grundsätzlich benutzbar bleiben soll
- JavaScript testen
- JavaScript-Code immer in mehreren Browsern testen
- Fehlerhafter JavaScript-Code kann im Gegensatz zu reinem (X)HTML Abstürze des Browsers oder des gesamten Systems verursachen
- Firefox bietet eine JavaScript-Konsole

Erstes JavaScript-Programm

JavaScript einbinden

script-Element

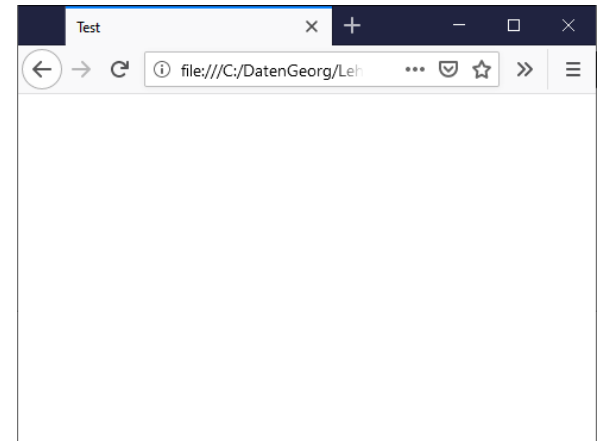
```
<body>
  <script> ....
    JavaScript-Anweisungen .....
  </script>
  ...
</body>
```

Erstes JavaScript-Programm

JavaScript einbinden

noscript-Element

```
<body>
  <script>
    alert ("JavaScript ist auf diesem Computer aktiviert!")
  </script>
  <noscript>
    <p>Diese Website ben&ouml;tigt JavaScript</p>
  </noscript>
  ...
</body>
```



Externe JavaScript-Datei

- Skripte können alternativ in einer externen Datei stehen (üblicherweise mit Endung `.js`)
- Wird wie folgt eingebunden:

```
<script src="ext.js"></script>
```

Wo können Skripte stehen?

- An einer beliebigen Stelle innerhalb eines (X)HTML-Dokuments
- Skripte werden ausgeführt, sobald das (X)HTML-Dokument geladen ist.
- Deklarierte Funktionen sollten innerhalb des `head`-Befehls stehen. Sie werden ausgeführt, wenn die entsprechende Funktion aufgerufen wird.

JavaScript Beispiele

Verändern von Inhalt und Layout

```
<!DOCTYPE html>
<html lang="de">
<head>
<title>Test</title>
</head>
<body>
  <h2>HTML-Inhalte verändern</h2>
  <p id="absatz">Ursprünglicher Text</p>
  <button type="button" onclick=
    "document.getElementById('absatz').innerHTML=
      'Veränderter Inhalt'">Inhalt ändern</button>
</body>
</html>
```



JavaScript

Sprachelemente

Kommentare

// Ein einzeiliger Kommentar

/* Kommentar über
mehrere Zeilen */

Use-strict-Befehl

- Verwendung von Verbesserungen und Neuerungen
- Strengere Regeln, daher robuster
- Aufgabe der Abwärtskompatibilität
- Verfügbar ab ECMAScript 5
- Erster Befehl im Skript (Ausnahme: Kommentare)

```
<script>  
    "use strict";  
    alert ("Hallo Welt!");  
</script>
```

JavaScript

Sprachelemente

Einfache Funktionen

- `alert()`;
Dient zur Ausgabe
- `prompt()`;
Dient zur Eingabe

```
<script>  
    "use strict";  
    alert ("Sie sind " + prompt ("Wie alt  
        sind Sie?") + " Jahre alt.");  
</script>
```

JavaScript

Sprachelemente

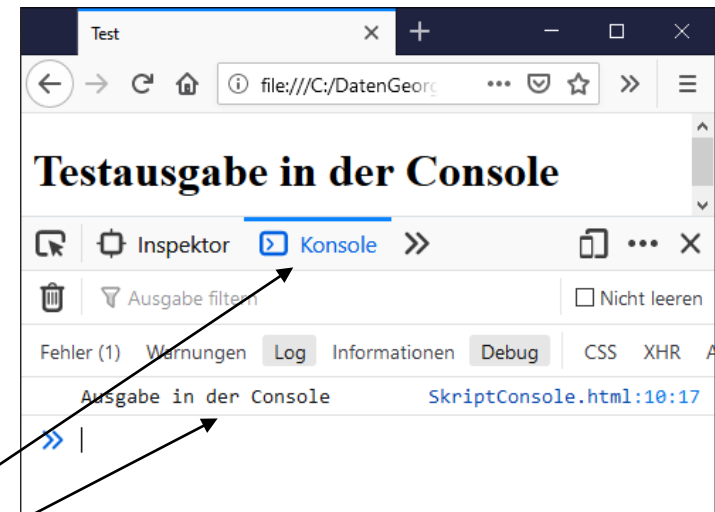
Variablen

- Werden mit dem Schlüsselwort `var`, `let` oder `const` deklariert
- Gültigkeit von `let` wie bei Java
- Gültigkeit von `var` ist die ganze Funktion
- `const` definiert Konstanten
- Variablennamen müssen mit einem Buchstaben oder Unterstrich (`_`) beginnen und unterscheiden Groß- und Kleinschreibung
- Deklarierte Variablen ohne Wertzuweisung besitzen den Wert `undefined`
- Globale Variablen können ohne Deklaration verwendet werden (implizit deklarierte Variablen; mit `use-strict` jedoch verboten)
- Variablentyp ergibt sich implizit aus dem zugewiesenen Wert
- Variable kann jederzeit einen neuen Wert und damit auch einen neuen Typ erhalten

JavaScript Sprachelemente

Ausgabe in der Webseite

- `document.write();`
- Schreibt den Inhalt in das Browserfenster.
- `document.writeln();`
- Schreibt den Inhalt in das Browserfenster und fügt einen Zeilenumbruch ein.
- `console.log();`
- Dient zu internen Ausgabe.



JavaScript

Ausgabe

Beispiele

```
document.write ("Doppelte Anführungszeichen");  
document.write ('Einfache Anführungszeichen');  
document.write ("Dies und " + "Das");  
document.write ("Dies und ", "Das");  
document.write ("Die Zahl ist " + zahl + "<br>");  
document.write (zahl + zahl);  
document.write ("<p onclick='info()'>" + zahl + zahl + "</p>");  
document.write (istGesund);
```

JavaScript

Typen

- Zahlen (intern nur Gleitpunktzahlen)
- Zeichenketten
- Boolean mit den Werten `true` und `false`

```
var alter = 33;
```

//Zahl

```
var first_name = "Marie";
```

//String

```
var surname = 'Risser';
```

//ebenfalls ein String

```
surname = 'Risser-Schmidt';
```

//Variable erhält neuen Wert

```
var istGesund = true;
```

//boolesche Variable

JavaScript

Sprachelemente

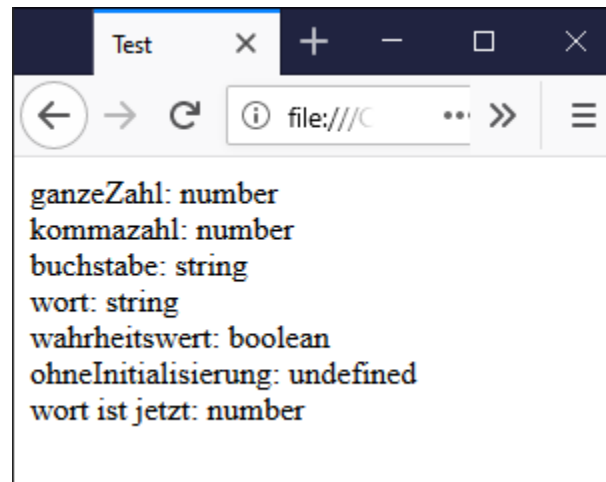
Typ einer Variable

```
<script>
  "use strict";
  let ganzeZahl = 3;
  let kommazahl = 2.34;
  let buchstabe = 'a';
  let wort = "Hallo";
  let wahrheitswert = true;
  let ohneInitialisierung;
  document.write ("ganzeZahl: " + typeof ganzeZahl + "<br>");
  document.write ("kommazahl: " + typeof kommazahl + "<br>");
  document.write ("buchstabe: " + typeof buchstabe + "<br>");
  document.write ("wort: " + typeof wort + "<br>");
  document.write ("wahrheitswert: " + typeof wahrheitswert + "<br>");
  document.write ("ohneInitialisierung: " +
    typeof ohneInitialisierung + "<br>");
  wort = 5;
  document.write ("wort ist jetzt: " +
    typeof wort + "<br>");
</script>
```


JavaScript

Sprachelemente

Typ einer Variable



The screenshot shows a web browser window with a tab titled 'Test'. The address bar displays 'file:///C:'. The developer console is open, showing the following JavaScript code and its corresponding data types:

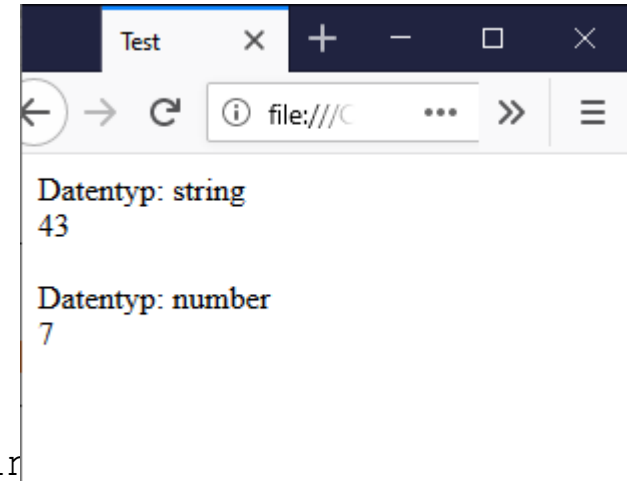
```
ganzeZahl: number  
kommazahl: number  
buchstabe: string  
wort: string  
wahrheitswert: boolean  
ohneInitialisierung: undefined  
wort ist jetzt: number
```

JavaScript Sprachelemente

Typkonvertierung

```
<script>
  "use strict";
  let eingabe = prompt ("Geben Sie eine Zahl ein");
  document.write ("Datentyp: " + typeof eingabe + "<br>");
  document.write (eingabe + 3);
  document.write ("<br><br>");
  eingabe = Number(eingabe);
  document.write ("Datentyp: " + typeof eingabe + "<br>");
  document.write (eingabe + 3);
</script>
```

- Der mit `prompt()` eingelesene Wert ist vom Typ `String`.
- Es wird die String-Konkatenation ausgeführt
- Nach der expliziten Typkonvertierung wird addiert.



JavaScript

Operatoren

Arithmetische Operatoren

- Übliche arithmetischen Operatoren: +, -, *, /
- Modulo-Operator %, der den Rest einer Division ermittelt
- ** für die Berechnung der Potenz

Verkettungsoperator

- Um Zeichenketten zu konkatenieren verwenden Sie in JavaScript das Zeichen „+“

JavaScript

Operatoren

Zusammengesetzte Zuweisungsoperatoren

- `a += b;` entspricht `a = a + b;`
- `a -= b;` entspricht `a = a - b;`
- ...
- `a += "beta";` entspricht `a = a + "beta";`

Kurzformen zum Inkrementieren und Dekrementieren

- `i++;` entspricht `i += 1;` bzw. `i = i + 1;`
- `i--;` entspricht `i -= 1;` bzw. `i = i - 1;`

Beispiele für Anweisungen

- `fahrenheit = celsius * (9/5) + 32;`
- `bmi = weight / (height * height);`

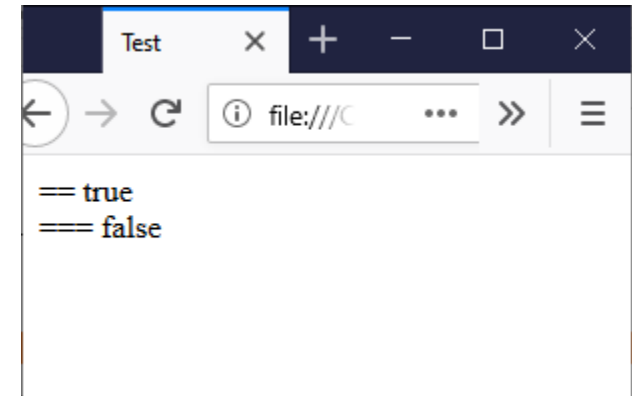
JavaScript Operatoren

Vergleichsoperatoren

- ==, !=, <, <=, >, >=
- Der Operator === prüft, ob Wert und Typ gleich sind.
- Analog gibt es einen Operator !==

Beispiel

```
<script>
  "use strict";
  let zahl = 5;
  let ziffer = "5";
  document.writeln ("== " + (zahl == ziffer) + "<br>");
  document.writeln ("=== " + (zahl === ziffer) + "<br>");
</script>
```



JavaScript

Operatoren

logische Operatoren

- `a || b` Logisches „Oder“
- `a && b` Logisches „Und“
- `!a` Logisches „Nicht“

JavaScript

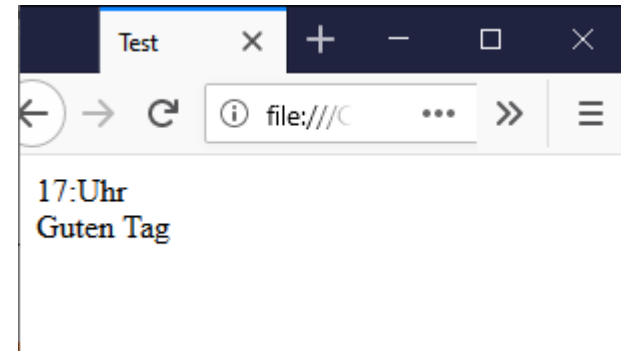
Kontrollstrukturen

if-Anweisung

Klasse `Date` liefert das aktuelle Datum zurückliefert

Operation `getHours()` ermittelt die jeweilige Stunde (0..23)

```
<script>
  "use strict";
  let date = new Date();
  let time = date.getHours();
  document.write(time + ":Uhr <br>");
  if (time > 18)
    {document.write("Es ist nach 18 Uhr <br>");}
  if (time < 12)
    {document.write("Es ist vor 12 Uhr <br>");}
  else
    {document.write("Guten Tag");}
</script>
```



JavaScript

Kontrollstrukturen

switch-Anweisung

Der Vergleich wird mit „==“ durchgeführt

```
<script>
  "use strict";
  let date = new Date();
  day = date.getDay(); //So=0, Mo=1, Di=2, Mi=3, Do=4, Fr=5, Sa=6
  switch (day)
  { case 6:
      document.write ("ein schönes Wochenende");
      break;
    case 0:
      document.write ("einen angenehmen Sonntag");
      break;
    default:
      document.write ("frohes Schaffen");  }
</script>
```


JavaScript

zusammengesetzte Datentypen

Felder

Definition: Datenstruktur, in der Datenelemente unter einem gemeinsamen Namen zusammengefasst werden (homogene Struktur).

Die Elemente des Array werden durch Indizierung des Arraynamens angesprochen.

Erzeugen von Feldern:

- `let meinArray = new Array();` //weniger gebräuchlich
- `let meinArray = [];`

JavaScript

zusammengesetzte Datentypen

Felder deklarieren

- `let meinArray = [];`
- `let names3 = ["Stefan", "Andreas", "Markus"];`

Auf Feldelemente zugreifen

- Mit Angabe der Indexposition. Beginn bei „0“.

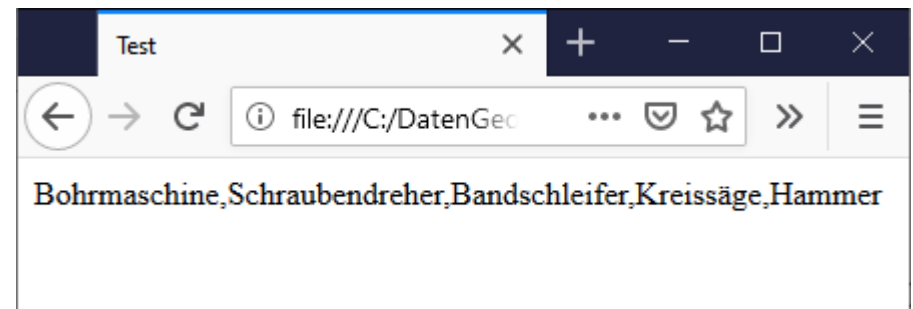
```
names3[0] = "Stefan";
```

JavaScript

zusammengesetzte Datentypen

Felder

```
<script>
  "use strict";
  let meinArray = ["Bohrmaschine", "Schraubendreher",
                  "Bandschleifer", "Kreissäge", "Hammer"];
  document.write(meinArray);
</script>
```



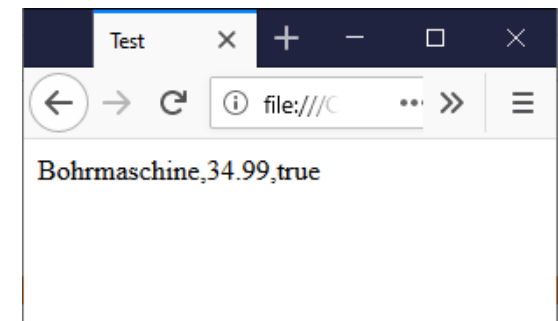
JavaScript

zusammengesetzte Datentypen

Felder

- Die Feldinhalte müssen nicht den gleichen Datentyp aufweisen

```
<script>  
  "use strict";  
  let meinArray = ["Bohrmaschine", 34.99, true];  
  document.write(meinArray);  
</script>
```



JavaScript

zusammengesetzte Datentypen

Dynamische Felder

- Neue Elemente dynamisch nach Bedarf erzeugen
- Elemente an Indexpositionen außerhalb der vorgegebenen Obergrenze neu definieren und ansprechen
- Nicht-definierte Elemente erhalten automatisch den Wert `undefined`
- `names3[4] = "Ina";` : neues Element
- Dazwischenliegende Elemente besitzen den Wert `undefined`

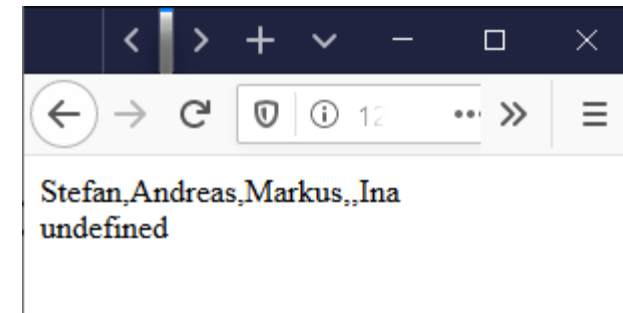
JavaScript

zusammengesetzte Datentypen

Felder

- Die Indexposition kann auch außerhalb der Obergrenze liegen (dynamische Felder)

```
<script>
  "use strict";
  let names3 = ["Stefan", "Andreas", "Markus"];
  names3[4] = "Ina";
  document.write(names3 + "<br>");
  document.write(typeof names3[3]);
</script>
```



JavaScript

zusammengesetzte Datentypen

Feld durchlaufen

- Eigenschaft `length`
- `names.length` liefert Größe des Feldes
- `names[length - 1]` liefert letztes Element des Feldes

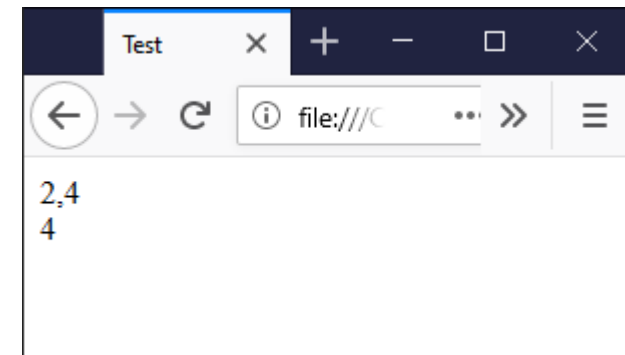
JavaScript

zusammengesetzte Datentypen

Mehrdimensionale Felder

- `let meinArray = [[2,4],[6,3],[8,9]];`

```
<script>  
  "use strict";  
  let meinArray = [[2,4],[6,3],[8,9]];  
  document.write(meinArray[0] + "<br>");  
  document.write(meinArray[0][1]);  
</script>
```



JavaScript

zusammengesetzte Datentypen

Mehrdimension

```
<script>
"use strict";
let meinArray = [];
meinArray[0] = ["Bohrmaschine", 34.99, true];
meinArray[1] = ["Schraubendreher", 4.99, true];
meinArray[2] = ["Bandschleifer", 41.99, false];
meinArray[3] = ["Kreissäge", 37.99, true];
meinArray[4] = ["Hammer", 6.99, false];
document.write(meinArray[0][1] + "<br>");
document.write(meinArray[2][2] + "<br>");
document.write(meinArray[4][0] + "<br>");
document.write(meinArray);
</script>
```



JavaScript

zusammengesetzte Datentypen

Map

Im Gegensatz zu Feldern können Elemente der Datenstruktur Map mit Hilfe eines Schlüsselbegriffs angesprochen werden. Der Vergleich erfolgt über „===“

Erzeugen einer Map:

- `let meineMap = new Map();`

Hinzufügen von Einträgen

- `meineMap.set("Produkttyp", "Bohrmaschine");`

Zugriff auf Elemente

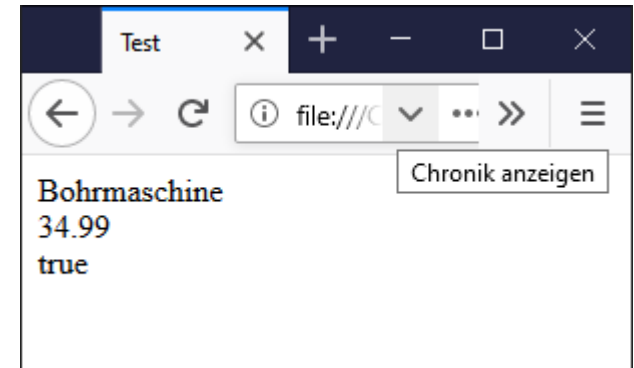
- `meineMap.get("Produkttyp");`

JavaScript

zusammengesetzte Datentypen

Map

```
<script>
  "use strict";
  let meineMap = new Map();
  meineMap.set("Produkttyp", "Bohrmaschine");
  meineMap.set("Preis", 34.99);
  meineMap.set("Verfuegbarkeit", true);
  document.write(meineMap.get("Produkttyp") + "<br>");
  document.write(meineMap.get("Preis") + "<br>");
  document.write(meineMap.get("Verfuegbarkeit") + "<br>");
</script>
```



JavaScript

zusammengesetzte Datentypen

Weitere Eigenschaften einer Map

`map.has(Schlüsselbegriff)`

- Gibt an, ob das entsprechende Feld enthalten ist

`map.delete(Schlüsselbegriff)`

- Löscht den entsprechenden Eintrag

`map.clear()`

- Löscht alle Einträge aus der Map

`map.size`

- Gibt die Anzahl der enthaltenen Felder zurück

JavaScript

zusammengesetzte Datentypen

Set

Der Datenstruktur Set stellt eine Menge im mathematischen Sinne dar.
Der Vergleich erfolgt über „===“

Erzeugen eines Set:

- `let meineSet = new Set();`

Hinzufügen von Einträgen

- `meineSet.add("Bohrmaschine");`

Zugriff auf Elemente

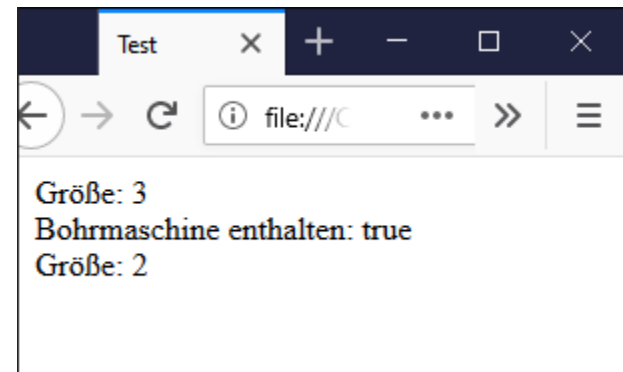
- `meineSet.has("Bohrmaschine");`

JavaScript

zusammengesetzte Datentypen

Set

```
<script>
  "use strict";
  let meinSet = new Set();
  meinSet.add("Bohrmaschine");
  meinSet.add("Bandschleifer");
  meinSet.add("Kreissäge");
  meinSet.add("Kreissäge");
  meinSet.add("Bohrmaschine");
  document.write("Größe: " + meinSet.size + "<br>");
  document.write("Bohrmaschine enthalten: " +
    meinSet.has("Bohrmaschine") + "<br>");
  meinSet.delete("Bohrmaschine");
  document.write("Größe: " + meinSet.size + "<br>");
</script>
```



JavaScript

zusammengesetzte Datentypen

Weitere Eigenschaften eines Set

```
let arr = ["Bohrmaschine", "Bandschleifer", "Kreissäge"];  
let meinSet = new Set(arr);
```

- Import aus einer anderen Datenstruktur

```
meinSet.clear();
```

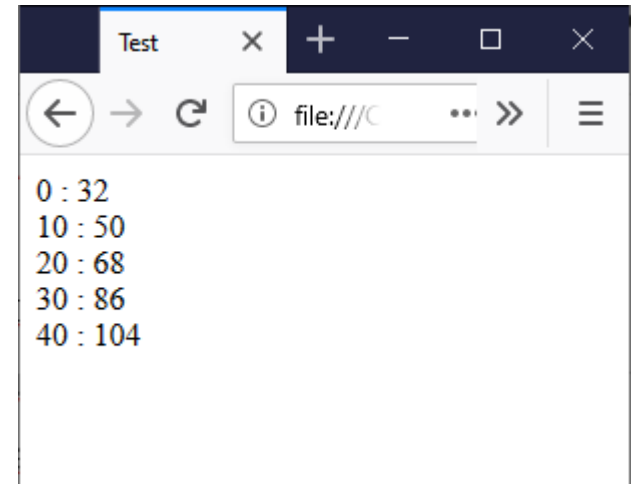
- Löschen aller Einträge

JavaScript

Kontrollstrukturen

While-Schleife

```
<script>
  "use strict";
  let celsius = 0;
  while (celsius <= 40){
    let fahrenheit = ((celsius * 9) / 5) + 32;
    document.write (celsius + " : " + fahrenheit + "<br>");
    celsius = celsius + 10;
  }
  celsius = 0;
</script>
```



JavaScript

Kontrollstrukturen

Do-While-Schleife

```
<script>
  "use strict";
  let celsius = 0;
  do {
    let fahrenheit = ((celsius * 9) / 5) + 32;
    document.write (celsius + " : " + fahrenheit + "<br>");
    celsius = celsius + 10;
  } while (celsius <= 40);
</script>
```

JavaScript

Kontrollstrukturen

For-Schleife

```
<script>
  "use strict";
  for (let celsius = 37; celsius <= 40; celsius++) {
    let fahrenheit = ((celsius * 9) / 5) + 32;
    document.write(celsius + " : " + fahrenheit + "<br>");
  }
</script>
```

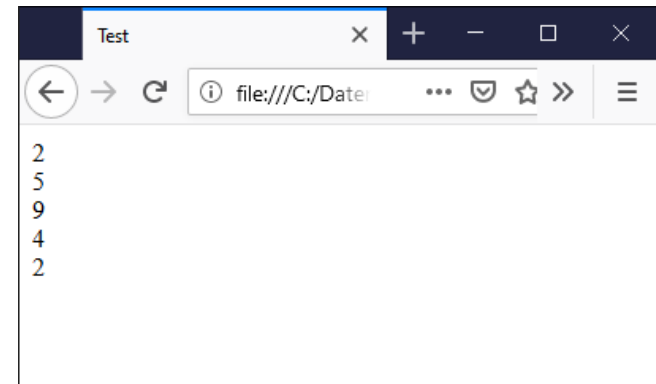
JavaScript

Kontrollstrukturen

For-of-Schleife

- Iteration durch die Datenstruktur.
- Jedes Element wird aufgezählt.
- Nur lesender Zugriff.

```
<script>  
  "use strict";  
  let arr = [2, 5, 9, 4, 2];  
  for (let wert of arr){  
    document.write(wert + "<br>");  
  }  
</script>
```



JavaScript

Kontrollstrukturen

Break, Continue

- Wie bei Java

```
<script>
  "use strict";
  let produkt = 1;
  let zahl, n;
  n = prompt("Wieviele Zahlen maximal einlesen? ");
  for (let i = 1; i <= n; i++) {
    zahl = prompt("naechste Zahl " + "(beenden mit -1): ");
    if (zahl == 0)          // ueberspringen
      continue;
    if (zahl == -1)        // Abbruch
      break;
    produkt *= zahl; }
  document.write("Produkt = " + produkt);
</script>
```

JavaScript

Funktionen

- Funktionen dienen dazu, Berechnungsschritte zusammenzufassen und im Dokument zur Verfügung zu stellen
- Funktionen beginnen mit dem Schlüsselwort `function`
- Danach folgen runde Klammern für die Funktionsargumente
- Danach folgen geschweifte Klammern für den Funktionsrumpf, der die Befehle enthält
- Die Funktion kann in beliebigen Stellen in der Webseite erfolgen
- JavaScript Funktionen können in einer externen Datei gespeichert werden und in die HTML-Seite eingebunden werden:
 - `<script src="quadrat.js"></script>`
- Funktionen werden im Allgemeinen im Kopf des HTML-Dokuments deklariert, damit sie im gesamten HTML-Dokument verwendet werden können

JavaScript

Funktionen

Beispiel

```
<script>  
  "use strict";  
  function begruessung() {  
    let name = prompt("Geben Sie Ihren Namen ein:");  
    alert("Herzlich willkommen, " + name); }  
  begruessung();  
</script>
```

Definition der Funktion

Aufruf der Funktion

JavaScript

Funktionen

Beispiel mit Funktion in externer Datei

```
<script src="funktion.js"></script>  
<script>  
    "use strict";  
    begruessung();  
</script>
```

JavaScript

Funktionen

Gültigkeitsbereich von Variablen

Lokale Variable

```
<script>
  "use strict";
  function begruessung() {
    let name = prompt("Geben Sie Ihren Namen ein:");
    alert("Herzlich willkommen, " + name); }
  begruessung();
  document.write("Ihr Name: " + name);
</script>
```

Hier nicht sichtbar

JavaScript

Funktionen

Gültigkeitsbereich von Variablen

```
<script>
  "use strict";
  function begruessung() {
    name = prompt("Geben Sie Ihren Namen ein:");
    alert("Herzlich willkommen, " + name); }
  let name;
  begruessung();
  document.write("Ihr Name: " + name);
</script>
```

Globale Variable

Hier sichtbar

JavaScript Funktionen

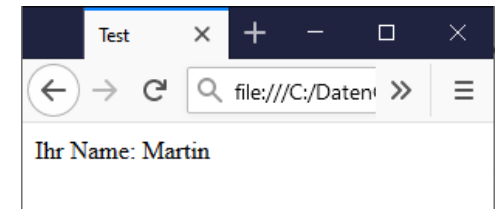
Lokale Variable verdeckt globale Variable

Gültigkeitsbereich von Variablen

Globale Variable

Lokale Variable

```
<script>
  "use strict";
  function begruessung() {
    let name = prompt("Geben Sie Ihren Namen ein:");
    alert("Herzlich willkommen, " + name); }
  let name = "Martin";
  begruessung();
  document.write("Ihr Name: " + name);
</script>
```



Hier ist die globale Variable sichtbar

JavaScript

Funktionen

Funktionen mit Argumenten

- Parameterübergabe erfolgt mittels "call by value"

```
function bmi2(weight, height)
{
  let bmi = weight / (height * height);
  let message = "BMI2: Der BMI für " + weight + " kg und "
               + height + " m ist " + bmi;
  alert (message);
}
```

Aufruf:

```
bmi2(40, 1.6);
```

JavaScript

Funktionen

Funktionen mit Argumenten

- Weiteres Beispiel

```
function begruessung(name, alter) {  
  document.write("Name: " + name + "<br>");  
  document.write("Alter: " + alter);  
}  
let anwender = prompt("Geben Sie Ihren Namen ein:");  
let alter = prompt("Geben Sie Ihr Alter ein:");  
begruessung(anwender, alter);
```

JavaScript

Funktionen

Funktionen mit Rückgabewert

- Beispiel

```
<script>
  "use strict";
  function bmi3(weight, height) {
    let bmi = weight / (height * height);
    return bmi;
  }
  let weight = prompt("Gewicht");
  let height = prompt("Größe");
  document.write("BMI: " + bmi3(weight, height));
</script>
```

JavaScript

Funktionen

Vorhandene Funktionen benutzen

- `parseInt()`
- Wandelt Zeichenkette in ganze Zahl

```
function istGanzZahl(zahl)
{
    if (parseInt(zahl) == zahl)
        alert("Ganze Zahl");
    else
        alert("Keine ganze Zahl");
}
```

JavaScript

Funktionen

Vorhandene Funktionen benutzen

- `parseFloat()`
- Analog

`isNaN()`: gibt `false` zurück, wenn die übergebene Variable eine Zahl oder leer ist,
sonst `true`

```
function istZahl(zahl)
{
    if (isNaN(zahl))
        alert("Keine Zahl");
    else
        alert("Zahl");
}
```

JavaScript

Objekte

Objekte deklarieren

- `Object`: Datentyp in JavaScript
- Ungeordnete Menge von Eigenschaften, von denen jede aus einem Namen und einem Wert besteht
- Objekte mit `new`-Operator deklarieren
- Wertzuweisung über die Eigenschaftsnamen

```
let member = new Object();  
member.number = 1234;  
member.name = "Michaela";  
let personName = member.name;
```


JavaScript

Objekte

Zugriff auf Objekte als „assoziatives Feld“

- Für Eigenschaften ist Punktoperator (Bezeichner!) und Operator [] (String!) definiert
- Assoziative Arrays assoziieren Werte mit Strings

```
member["number"] = 1234;
```

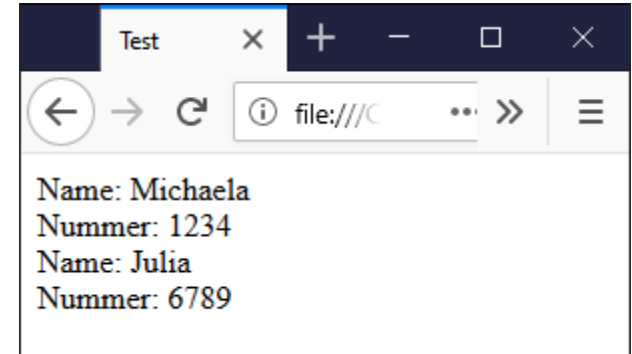
```
member["name"] = "Michaela";
```

```
var personName = member["name"];
```

JavaScript Objekte

Wahlweiser Zugriff auf Objekte

```
<script>
  "use strict";
  let member = new Object();
  member.number = 1234;
  member.name = "Michaela";
  let personName = member.name;
  let personNummer = member["number"];
  document.write("Name: " + personName + "<br>");
  document.write("Nummer: " + personNummer + "<br>");
  member["name"] = "Julia";
  member.number = 6789;
  document.write("Name: " + member.name + "<br>");
  document.write("Nummer: " + member["number"]);
</script>
```



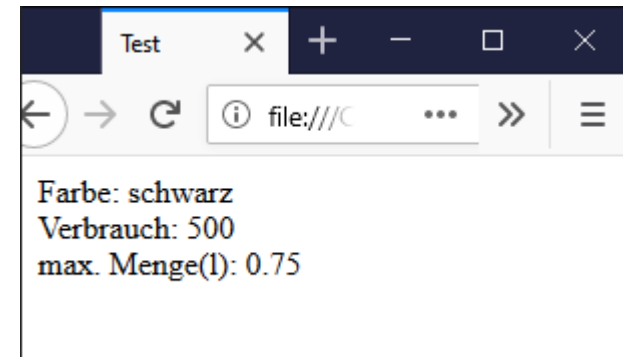
JavaScript

Kontrollstrukturen für Objekte

For-in-Schleife

- Iteration durch die Attribute einer Objektes

```
<script>
  "use strict";
  let meineKaffeemaschine = new Object();
  meineKaffeemaschine.Farbe = "schwarz";
  meineKaffeemaschine.Verbrauch = 500;
  meineKaffeemaschine["max. Menge(l)"] = 0.75;
  for (let i in meineKaffeemaschine) {
    document.write(i + ": " + meineKaffeemaschine[i] + "<br>");
  }
</script>
```



JavaScript Objekte

Methoden

- Feldelementen können auch Funktionen zugewiesen werden.

```
let meineKaffeemaschine = new Object();  
meineKaffeemaschine.Farbe = "schwarz";  
meineKaffeemaschine.Verbrauch = 500;  
meineKaffeemaschine["max. Menge (l)"] = 0.75;  
meineKaffeemaschine.Kochen=Kochen;  
function Kochen(minuten) {  
    document.write("Kaffe kocht: " + minuten+ "<br>");  
}  
meineKaffeemaschine.Kochen(10);  
meineKaffeemaschine["Kochen"](8);
```

JavaScript Objekte

Objekte deklarieren 2. Variante

- `function`: Konstruktor definieren
- Keine Deklaration im objektorientierten Sinn, sondern mit Konstruktor
- Konstruktor: Funktion, die neues Objekt erzeugt

```
function Mitarbeiter(nummer, nachname, bruttogehalt)
{
    //Attribute bzw. Eigenschaften
    this.personalnr = nummer;
    this.nachname = nachname;
    this.gehalt = bruttogehalt;
}
```

JavaScript Objekte

Methoden für Klasse definieren

- Zuweisung der Methode über den Namen

```
function Mitarbeiter(nummer, nachname, bruttogehalt)
{
    //Eigenschaften bzw. Attribute
    . . .
    //Methoden
    this.erhoeheGehalt = Gehaltserhoehung;
    this.ausgabe = ausgabe;
    this.setGehalt = setGehalt;
}
```

JavaScript Objekte

Methoden für Klasse definieren

- Implementierung der Methoden

```
function Gehaltserhoehung(erhoehung)
{  this.gehalt = this.gehalt + erhoehung;  }
```

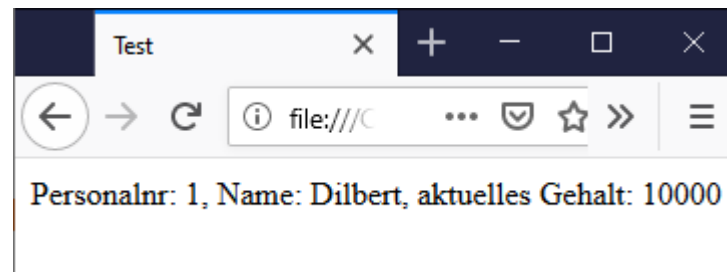
```
function ausgabe()
{  return ("Personalnr: " + this.personalnr
          + ", Name: " + this.nachname
          + ", aktuelles Gehalt: " + this.gehalt
          + "<br>"); }
```

```
function setGehalt(gehalt)
{  this.gehalt = gehalt; }
```

JavaScript Objekte

Beispiel

```
let dilbert = new Mitarbeiter(1, "Dilbert", 10000);  
document.write(dilbert.ausgabe());
```



JavaScript Objekte

Objekte deklarieren 3. Variante (seit 2015)

- `class`: Definition der Klasse
- `constructor`: Name des Konstruktors der Klasse
- Keine Zugriffsrechte; private Attribute beginnen mit „_“, per Konvention

```
class Mitarbeiter{  
    constructor(nummer, nachname, bruttogehalt){  
        this.personalnr = nummer;  
        this.nachname = nachname;  
        this.gehalt = bruttogehalt;  
    //Methoden  
}
```

JavaScript

Objekte

Objekte deklarieren 3. Variante (seit 2015)

```
class Mitarbeiter{
  constructor(nummer, nachname, bruttogehalt){
    this.personalnr = nummer;
    this.nachname = nachname;
    this.gehalt = bruttogehalt;}

  gehaltserhoehung(erhoehung){
    this.gehalt = this.gehalt + erhoehung;}

  ausgabe() {
    return ("Personalnr: " + this.personalnr + ", Name: " +
    this.nachname + ", aktuelles Gehalt: " + this.gehalt + "<br>");}

  setGehalt(gehalt){this.gehalt = gehalt; }
}
```

JavaScript Objekte

Beispiel

```
let dilbert = new Mitarbeiter(1, "Dilbert", 10000);  
document.write(dilbert.ausgabe());
```



JavaScript

Existierende Klassen

Klassen verwenden

JavaScript-Klasse Date

- `date = new Date();`
Erzeugt ein Objekt mit aktuellem Datum und aktueller Zeit
- `date.getHours()`
Ermittelt die Stunde
- `date.getDate()`
Ermittelt den Tag des Monats
- `date.getMonth()`
Ermittelt den Monat, wobei gilt: Jan=0, Feb=1, ... , Dez=11
- `date.getDay()`
Ermittelt den Wochentag, wobei gilt: So=0, Mo=1, Di=2, Mi=3, Do=4, Fr=5, Sa=6.

JavaScript

Existierende Klassen

Klassen verwenden

JavaScript-Klasse `String`

- Einfache Zeichenketten bzw. Strings: `let name1 = "Marie";`
- String-Objekt: `let name1 = new String ("Marie");`
- Automatische Konvertierung zwischen beiden Datentypen!
- `nachname.length`: Anzahl Zeichen ermitteln
- `i = nachname.indexOf('a')`: Position von 'a' (alternativ Teilstring!) im String, sonst -1
- `nachname.charAt(length-1)`: Letztes Zeichen lesen

```
var s1 = new String("Scheiben");  
var s2 = new String("wischer");  
s1 = s1 + s2;
```

JavaScript Objektmodelle

BOM (Browser Object Model)

- Stellt das `window` Objekt zur Verfügung.
- `alert()` und `prompt()` sind Methoden des `window` Objektes. Ausgeschrieben heißt es `window.alert("Hallo");`

DOM (Document Object Model)

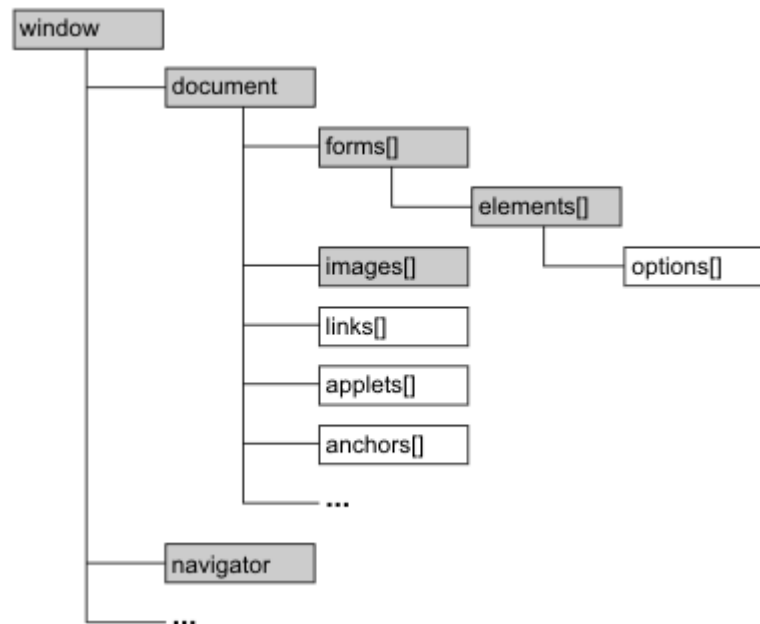
- Das DOM ist vom BOM abgeleitet und betrifft das Objekt `document`. Verwendet wurden bereits Methoden, wie `document.write()`; Ausgeschrieben heißt es `window.document.write("Hallo");`

CSSOM (CSS Object Model)

- Das CSSOM bietet Zugriff auf die Layouteigenschaften.

Objektmodell

DOM stellt Objekte, Eigenschaften und Methoden zur Verfügung



JavaScript Events

Behandlung von Ereignissen

- Der Browser kann auf Ereignisse reagieren.
- Ein Ereignis kann eine Benutzerinteraktion sein, wie z.B. ein Mausklick. Es können aber auch zeitgesteuerte Ereignisse sein.
- Das Ereignis enthält nähere Informationen über sich.
- Die Reaktion auf ein Ereignis kann der Aufruf einer JavaScript Funktion sein.
- Diese Funktion kann dann beispielsweise auf den Inhalt der Webseite zugreifen (über die DOM API) und diesen manipulieren.

Übersicht über alle Events: <https://developer.mozilla.org/de/docs/Web/Events>

JavaScript Events

Auswahl

- `onload`: Wird aufgerufen, wenn das Dokument vollständig geladen ist.
- `onunload`: Wird aufgerufen, wenn der Browser das aktuelle Dokument verlässt.
- `onclick`
- `ondblclick`
- `onmouseover`
- `onmouseout`
- `onmousedown`
- `onmouseup`
- `onsubmit`: Wird unmittelbar vor dem Senden des Formularinhalts aufgerufen. Gibt eine Prüffunktion `false` zurück, dann wird der Formularinhalt nicht abgesendet.
- `onreset`: Wird unmittelbar vor dem Rücksetzen der Formularinhalte aufgerufen. Gibt eine Prüffunktion `false` zurück, dann erfolgt kein Rücksetzen.

JavaScript Events

Reagieren auf Ereignisse

- Es müssen Eventhandler für die Elemente registriert werden, die auf die Events reagieren sollen.
- Die Eventhandler verbinden Ereignis und JavaScript Funktionen.

Beispiel

```
<input value="Hier klicken!"  
  onclick="alert('Sie haben den Button geklickt!')"  
  type="button">
```

JavaScript Events

Beispiel

```
<body>
  <input value="Hier klicken!"
    onclick="verdopplung()" type="button">
  <script>
    "use strict";
    function verdopplung() {
      let zahl = prompt("Geben Sie eine Zahl ein:");
      zahl *= 2;
      alert("Doppelter Wert: " + zahl);
    }
  </script>
</body>
```

- Eventhandler können den Namen einer bereits existierenden JavaScript Funktion angeben. Diese Funktion wird beim Eintreffen des Ereignisses aufgerufen.

JavaScript

Events

Weiteres Beispiel

```
<body>
  <p>Absatz 1</p>
  <p id="absatz">Absatz 2</p>
  <p>Absatz 3</p>
  <script> absatz.onmouseover = function () {
    alert("Hier befindet sich Absatz 2.");
  }
</script>
</body>
```

- Ist eine `id` vergeben, kann sie für den Eventhandler genutzt werden.
- Es können auch vorhandene Funktionen in dieser genutzt werden:

`absatz.onmouseover = nachricht;` Nachricht ist dabei eine JavaScript Funktion:
`function nachricht(){...}` Diese wird dabei ohne runde Klammern angegeben.

JavaScript

Events

Flexible Eventbearbeitung

- Bisher kann nur ein Event für ein Element definiert werden.
- Mehrere gleiche Eventhandler für dasselbe Element überschreiben sich gegenseitig.
- Zum flexibleren Bearbeiten mehrerer Events ist ein Eventlistener nötig.
- Eventlistener können im Nachhinein für Elemente registriert werden und können mehrere Events abfangen und bearbeiten: `addEventListener()`
- Eventlistener können im Nachhinein von Elementen entfernt werden: `removeEventListener()`
- **Die Arbeit mit dem Eventlistener ist zu bevorzugen, da somit Inhalt und Verhalten getrennt sind.**

JavaScript

Events

Beispiel

```
<body>
  <p>Absatz 1</p>
  <p id="absatz">Absatz 2</p>
  <p>Absatz 3</p>
  <script> function nachricht1() {
    alert("Hier befindet sich Absatz 2.");
  }
    function nachricht2() {
    alert("Auf Wiedersehen.");
  }
  absatz.addEventListener("mouseover", nachricht1);
  absatz.addEventListener("mouseout", nachricht2);

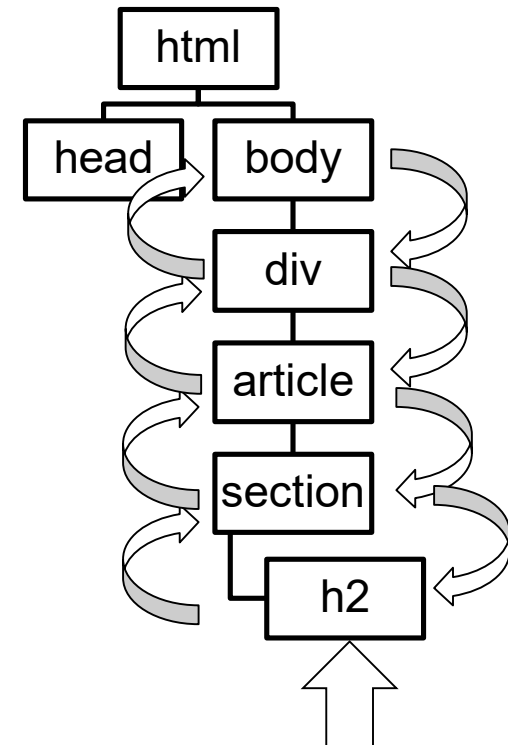
</script>
</body>
```

JavaScript Events

Flexible Propagierung

- Sind mehrere Eventhandler an ineinander geschachtelten Elementen registriert, muss entschieden werden, welcher davon angesprochen wird:
 - Trifft das Event nur am spezifischsten Element auf (Target)?
 - Läuft das Event entlang des DOM Baumes von der Wurzel zum Element (Capturing)?
 - Läuft das Event entlang des DOM Baumes vom Element zur Wurzel (Bubbling)?

Beispiel:



JavaScript

Events

Beispiel Event Bubbling

```
<body id="b">
  <div id="d">Das ist ein div-Element
    <p id="p"> Hier steht ein Text, bei dem ein Teil
      <strong id="s">fett</strong> gedruckt ist.</p>
    </div>
  <script> function bodyTag() {alert("Body-Tag"); }
    b.onclick = bodyTag;
    function divTag() {alert("Div-Tag");}
    d.onclick = divTag;
    function pTag() {alert("P-Tag");}
    p.onclick = pTag;
    function strongTag() {alert("Strong-Tag");}
    s.onclick = strongTag;
  </script>
</body>
```


JavaScript Events

Flexible Eventbearbeitung

- Das Verhalten wird Event Bubbling genannt. Das Event läuft vom spezifischsten Element über alle Vorfahren bis hin zur Wurzel.
- Die Verhalten Event Capturing oder Target sind auch möglich.
- Wird der Befehl `event.stopPropagation()` eingefügt, wird die Weiterleitung des Events beendet.
- `event.target` bezeichnet das (spezifischste) Element, bei dem das Event aufgetroffen ist.
- Mit `event.target.closest('p')`; kann man den nächsten Vorfahren im DOM Baum ansprechen, der ein `p` Element ist.
- In den Funktionen kann das Schlüsselwort `this` verwandt werden. Es bezieht sich auf das Element, das die Funktion aufgerufen hat.

JavaScript Events

Beispiel Event Capturing

```
<body id="b">
  <div id="d">Das ist ein div-Element
    <p id="p"> Hier steht ein Text, bei dem ein Teil
      <strong id="s">fett</strong> gedruckt ist. </p> </div>
<script> function bodyTag() {
  alert("Body-Tag");      }
document.getElementById('b').addEventListener('click', bodyTag, true);
function divTag() {
  alert("Div-Tag");      }
document.getElementById('d').addEventListener('click', divTag, true);
function pTag() {
  alert("P-Tag");        }
document.getElementById('p').addEventListener('click', pTag, true);
function strongTag() {
  alert("Strong-Tag");    }
document.getElementById('s').addEventListener('click', strongTag, true);
</script> </body>
```

JavaScript

Events

Beispiel Events delegieren

```
<body id="b">
  <p>Absatz 1</p>
  <p>Absatz 2</p>
  <p>Absatz 3</p>
  <script> function hintergrund() {
    event.target.style.backgroundColor = 'green';
  }
  b.onclick = hintergrund;
</script>
</body>
```

- Das Event wird von dem übergeordneten Element behandelt und es wird nur ein Eventhandler benötigt.
- Durch die Verwendung von `target` kann das Element angesprochen werden, bei dem das Event aufgetroffen ist.

JavaScript

Events

Beispiel Events delegieren

```
<body id="b">
  <p>Absatz 1</p>
  <p>Absatz 2</p>
  <p>Absatz 3 mit <strong>fett</strong> gedrucktem Wort</p>
  <script> function hintergrund() {
    let p = event.target.closest('p');
    p.style.backgroundColor = 'green';
  }
  b.onclick = hintergrund;
</script>
</body>
```

- Wenn das Event beim Element `strong` auftritt wird als Ziel der im DOM Baum nächste Vorfahr `p` als Ziel für die Bearbeitung gesetzt.

JavaScript

Objekt `window`

Globales Objekt für clientseitiges JavaScript

Repräsentiert das Browser-Fenster

Operationen

- `alert()`: zeigt Text in Mitteilungsfenster an
- `confirm()`: zeigt Text in Bestätigungsfenster an

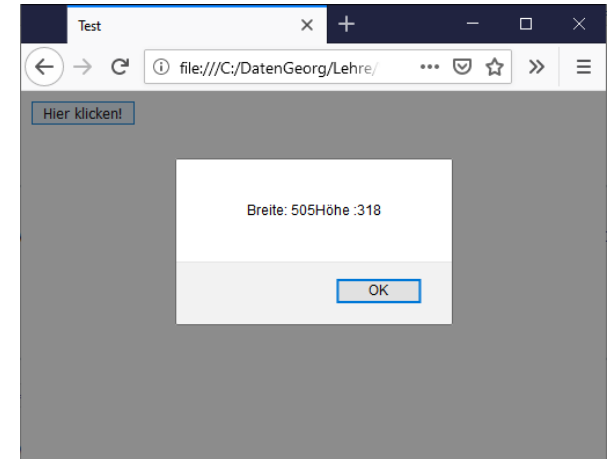
```
alert ("Der Server wird in 5 Minuten heruntergefahren");
```

```
erg = confirm ("Zum Löschen wollen klicken Sie auf OK");
```

Attribute

- `innerWidth`: Breite des Fenster
- `innerHeight`: Höhe des Fensters

```
<body>
  <input value="Hier klicken!"
    onclick="dimensionenAnzeigen()" type="button">
  <script>
    "use strict";
    function dimensionenAnzeigen() {
      alert("Breite: " + window.innerWidth + " Höhe : " +
        window.innerHeight); }
  </script>
</body>
```



Weitere Eigenschaften: <https://wiki.selfhtml.org/wiki/JavaScript/Window>

JavaScript

Objekt document

Wurzel des „Document Tree“

Globales Objekt, um auf das HTML Dokument zuzugreifen.

`write()`: Fügt einen oder mehrere Strings an das aktuell geöffnete HTML-Dokument an.

```
document.write ("Ideal ist ein BMI zwischen 20 und 24");  
document.write ("Ihr Gewicht von " + weight  
                + " kg und Ihre Größe von " + height  
                + " m ergeben einen BMI von " + bmi);  
document.write ("Mein Name ist", "Lisa");
```

JavaScript

Objekt document

- Es kann über den Namen auf Elemente des Dokumentbaumes zugegriffen werden.:
`document.head, document.body`
- Es kann über die Struktur des DOM Trees auf Elemente zugegriffen werden:
`document.body.firstChild,`
`document.body.firstChild.nextElementSibling,`
`document.body.firstChild.nextElementSibling.nextElementSibling.firstChild.nextElementSibling`
- Zugriff auf ein Element des HTML-Dokuments über das Universalattribut `id`:
`document.getElementById()`
- Zugriff auf ein Feld mit allen Elementen des Dokuments über Namen oder Klasse:
`document.getElementsByName(), document.getElementsByTagName()`

JavaScript

Objekt document

Beispiele

```
<body>
  <h1 id="Ueberschrift1">Überschrift 1</h1>
  <h2 class="cool">Überschrift 2</h2>
  <p>Absatz mit <i>einem kursiven Bereich</i> und einem
    <strong>fett</strong> gedruckten Wort.</p>
  <h2>Überschrift 2</h2>
<p>Weiterer Absatz</p>
<script>
  document.open();
  document.write("<div>" + document.head + "</div>");
  document.write("<div>" + document.body + "</div>");
  document.write("<div>" + document.body.firstChild + "</div>");
  document.write("<div>" +
    document.body.firstChild.nextElementSibling + "</div>");
```

JavaScript

Objekt document

Beispiele

```
document.write("<div>" +  
  document.body.firstChild.nextElementSibling.  
  nextElementSibling.firstChild.nextElementSibling +  
  "</div>");  
document.write("<div>" + document.getElementById("Ueberschrift1") +  
  "</div>");  
document.write("<div>" + document.getElementsByTagName("h2")[0] +  
  "</div>");  
document.write("<div>" + document.getElementsByClassName("cool")[0]  
+ "</div>");  
</script>  
</body>
```

JavaScript

Objekt document

Zugriff auf den HTML Inhalt eines Elementes

- `innerHTML` greift auf den Inhalt des Elementes zu. Es dürfen HTML Elemente enthalten sein.
- Dies kann sowohl lesend, als auch schreibend verwendet werden.

- **Beispiele:**

```
document.getElementById("Ueberschrift1").innerHTML;  
document.getElementsByTagName("h2")[0].innerHTML;  
document.getElementsByClassName("cool")[0].innerHTML;
```

oder

```
document.getElementById("Ueberschrift1").innerHTML = "Neue Welt";
```

JavaScript

Objekt document

Ohne innerHTML

Test

file:///C:/DatenGeorg/Lehre/VorlesungWebt

Überschrift 1

Überschrift 2

Absatz mit *einem kursiven Bereich* und einem **fett** gedruckten Wort.

Überschrift 2

Weiterer Absatz

- [object HTMLHeadElement]
- [object HTMLBodyElement]
- [object HTMLHeadingElement]
- [object HTMLHeadingElement]
- [object HTMLDivElement]
- [object HTMLHeadingElement]
- [object HTMLHeadingElement]
- [object HTMLHeadingElement]

Mit innerHTML

Test

file:///C:/DatenGeorg/Lehre/VorlesungWebt

Überschrift 1

Überschrift 2

Absatz mit *einem kursiven Bereich* und einem **fett** gedruckten Wort.

Überschrift 2

Weiterer Absatz

- Überschrift 1
- Überschrift 2
- fett
- Überschrift 1
- Überschrift 2
- Überschrift 2

JavaScript

Objekt document

Zugriff auf weitere Eigenschaften

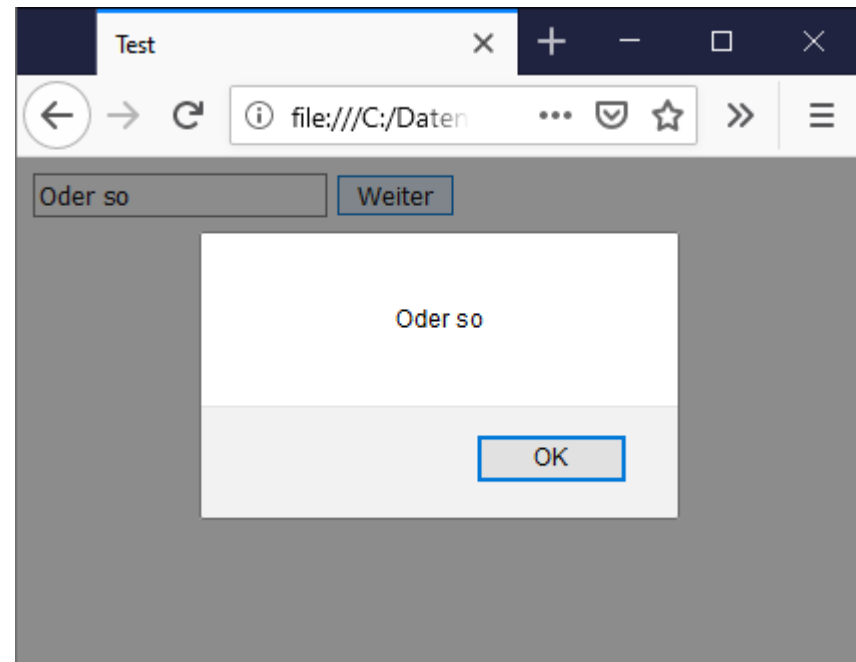
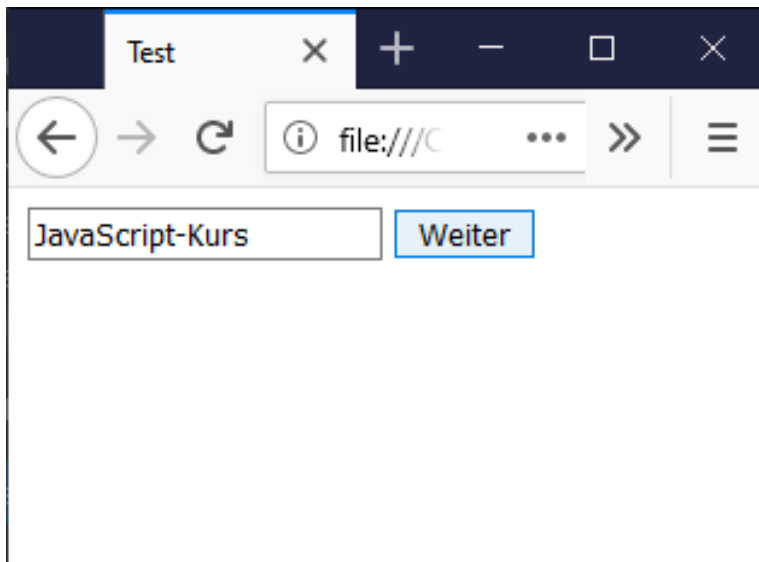
- Es kann auf Formularinhalte lesend oder schreibend zugegriffen werden über das Attribut: `value`

```
<body>
  <input id = "eingabefeld">
  <input value="Weiter" onclick="auslesen()" type="button">
<script>
document.getElementById("eingabefeld").value = "JavaScript-Kurs";
function auslesen(){
  let inhalt = document.getElementById("eingabefeld").value;
  alert(inhalt); }
</script>
</body>
```

JavaScript

Objekt document

Beispiel



Liste der Eigenschaften <https://wiki.selfhtml.org/wiki/JavaScript/DOM/Element>

JavaScript

Objekt document

Zugriff auf weitere Eigenschaften

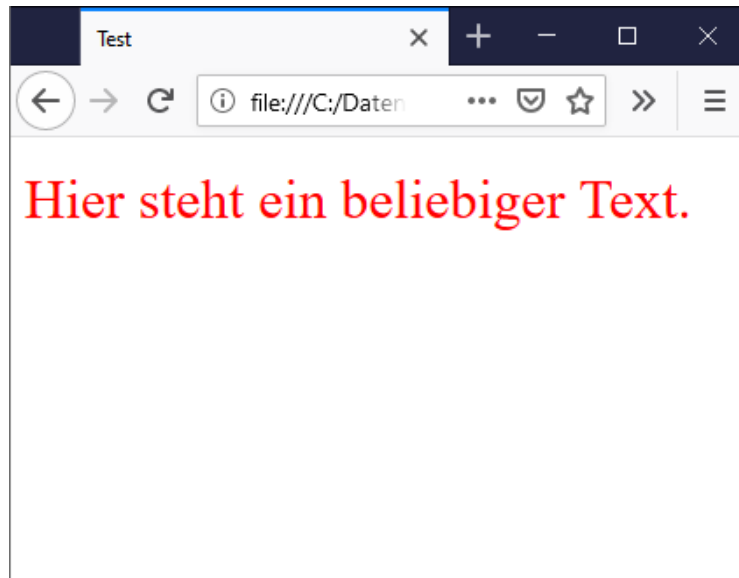
- Es kann auf Style Eigenschaften zugegriffen werden.

```
<body>
  <p>
    <font id="schrift">Hier steht ein beliebiger Text.</font>
  </p>
  <script>
    document.getElementById("schrift").size = "6";
    document.getElementById("schrift").color = "red";
  </script>
</body>
```

JavaScript

Objekt document

Beispiel



Liste der Eigenschaften

<https://wiki.selfhtml.org/wiki/JavaScript/DOM/Element/style>

https://www.w3schools.com/jsref/dom_obj_style.asp

JavaScript

Objekt `style`

Style

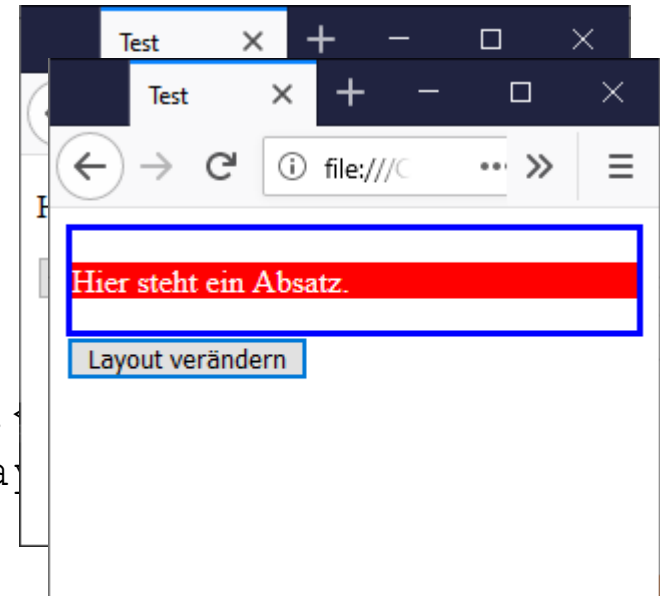
- Gehört zu einem Element der Seite
- Zugriff `Element.style.eigenschaft = 'wert';`

Vgl.: <https://wiki.selfhtml.org/wiki/JavaScript/DOM/Element/style>

Beispiel

JavaScript Objekt style

```
<body>
  <div id="div">
    <p id="absatz">Hier steht ein Absatz.
  </div> <button onclick="hintergrund()">Layout verändern
<script>
  function hintergrund() {
    document.getElementById("absatz").style.background = "red";
    document.getElementById("absatz").style.fontSize = 30;
    document.getElementById("absatz").style.color = "white";
    document.getElementById("div").style.width = 150;
    document.getElementById("div").style.border = "3px solid blue";
  }
</script>
</body>
```



JavaScript

Feld von Bildern `images[]`

Image-Objekt: jedes Bild eines HTML-Dokuments

- `images[]`: alle Bilder des Dokuments
- Erstes Bild des Dokuments: `document.images[0]`
- Zugriff über Bild-ID: `document.images.myImage`

Attribut:

`src`: Name der Bilddatei

Vgl.: <https://developer.mozilla.org/en-US/docs/Web/API/Document/images>
<https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement>

JavaScript

Feld von Bildern `images[]`

Beispiel

```
farbe = new Image();
farbe.height = 210;
farbe.width = 300;
farbe.src = "images/jogging.jpg";

function farbbild(){
    var id=document.getElementById("pic");
    id.src=farbe.src;}

sw = new Image(300, 210);
sw.src = "images/jogging_sw.jpg";

function sw_bild(){
    var id=document.getElementById("pic");
    id.src=sw.src;}

...

<br>
<input type="button" value="Farbe" onclick="farbbild()">
<input type="button" value="Schwarzweiß" onclick="sw_bild()">
```

JavaScript

Feld von Formularen `forms[]`

Form-Objekt: jedes Formular im HTML-Dokument

- `forms[]`: alle Formulare eines Dokuments

Zugriff:

- `document.forms[0]`: 1. Formular im Dokument
- `document.forms[1]`: 2. Formular im Dokument
- `document.forms.formular`: Formular mit dem Namen `formular` im Dokument
- Inhalt der Formulare sind Strings. Zur Bearbeitung von Strings vgl.:
https://www.w3schools.com/jsref/jsref_obj_string.asp

JavaScript

Feld von Formularen `forms []`

`elements []`: enthält Formularfelder eines Formulars

- `document.forms[0].elements[0]`: 1. Element im 1. Formular des Dokuments
- `document.forms.formular.elements.feld`: Element mit dem Namen `feld` im Formular mit dem Namen `formular`

Folgende Zugriffe sind möglich:

`document.forms[0].elements[0].eigenschaft`:

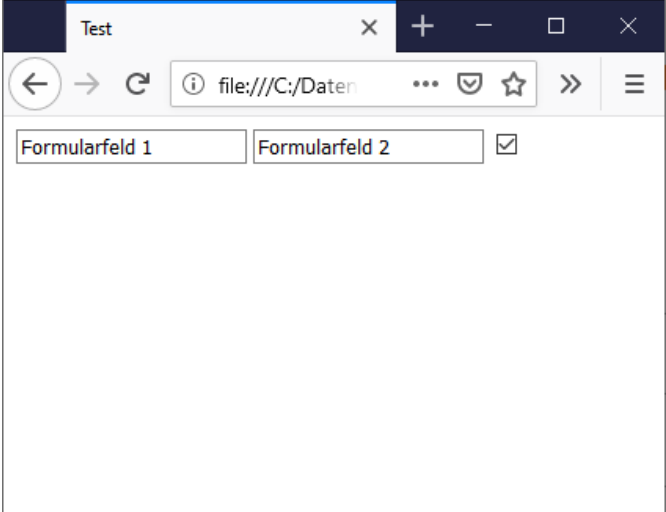
Beispiel:

- `document.forms[0].elements[0].value`
Aktueller Wert des Elements, der durch den Benutzer eingegeben wird.

JavaScript Formulare

Zugriff auf Inhalte

- Kurzschreibweise



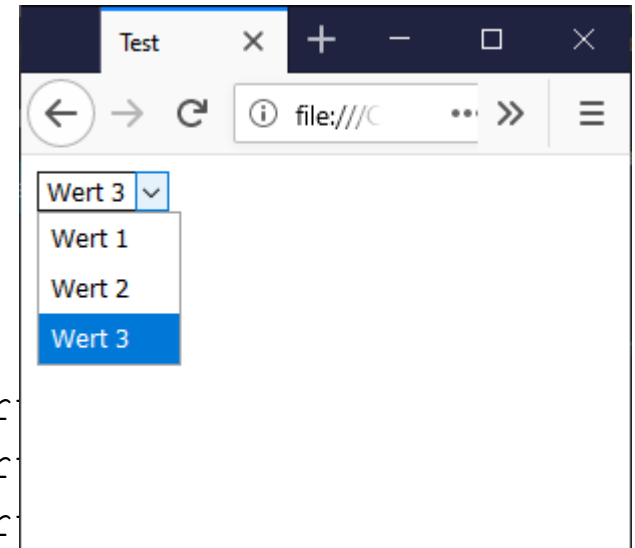
The screenshot shows a web browser window with the title 'Test'. The address bar displays 'file:///C:/Daten'. The page content includes two text input fields labeled 'Formularfeld 1' and 'Formularfeld 2', followed by a checked checkbox.

```
<body>
  <form name="formular">
    <input name="feld1" value="Formularfeld 1">
    <input name="feld2" value="Formularfeld 2">
    <input name="feld3" type="checkbox">
  </form>
  <script>
    formular.feld3.checked = true;
  </script>
</body>
```

JavaScript Formulare

Weiteres Beispiel

```
<body>
  <form>
    <select id="auswahl">
      <option value="auswahl1">Wer
      <option value="auswahl2">Wer
      <option value="auswahl3">Wer
    </select>
  </form>
  <script>
    auswahl.selectedIndex = 2;
    auswahl.options[2].selected = true; //gleiche Bedeutung
    auswahl.value = "auswahl3";        //gleiche Bedeutung
  </script>
</body>
```



JavaScript

Events für Formulare

Bearbeitung von Formularen

- `focus()` Der Focus wird auf dieses Element gesetzt.
- `onblur` Der Focus wird auf ein anderes Element gesetzt.
- `onchange` Der Wert eines Elementes hat sich geändert, z.B. Checkboxes und Radio Buttons
- `onsubmit` Der `submit` Knopf wurde gedrückt. Rückgabewert `true`: Versenden des Formulars
- `onreset` Der `reset` Button wurde gedrückt. Rückgabewert `true`: Löschen des Formulars
- `onclick` Der Button wurde gedrückt. (Vgl. Beispiel zu `document`)

JavaScript

Formulare

Beispiel

```
<body>
  <form name="formular">
    <input name="vorname" value="Vorname">
    <input name="nachname" value="Nachname">
    <input name="check" type="checkbox">
  </form>
  <script>
    formular.nachname.focus(); //Feld Nachname selektiert
    formular.vorname.onfocus = hilfVorname;
    formular.vorname.onblur = checkVorname;
    formular.vorname.oninput = checkZeichen;
    //->
```

JavaScript

Formulare

Beispiel

```
let angezeigt = false;
function hilfVorname() {
    if (!angezeigt) {
        alert("Bitte Vornamen eingeben");
        angezeigt = true;    }    }

function checkZeichen() {
    if (formular.vorname.value.indexOf("@") != -1) {
        alert("Unerlaubtes Zeichen");    }    }

function checkVorname() {
    if (formular.vorname.value == "") {
        alert("Bitte Vornamen eingeben");    }    }

</script>
</body>
```

JavaScript Formulare

Beispiel

```
<body>
  <form action="foo()" onsubmit="return check(this)"
    onreset="return confirm('Mit Ok Formulardaten löschen')">
    <p>Inhalt <input type="text" name="beliebig"> </p>
    <p><input type="submit" value="Prüfen">
      <input type="reset" value="Abbruch"> </p>
  </form>

  <script>
    function check(self) {
      if (self.bbeliebig.value == "") {
        alert("Bitte Daten eingeben");
        return false; }
      return true; }
  </script>
</body>
```

JavaScript

Weitere Objekte

Location

- `window.document.location`: Zugriff auf diverse Eigenschaften des Dokumentes
- `location.href`: URI des Dokumentes1. Formular im Dokument
- `location.search`: Suchparameter in der URI bei der GET Methode
- `location.host`: Hostname
- `location.protocol`: Verwendetes Protokoll
- `location.reload()` : Lädt die Seite neu

JavaScript

Location

Beispiel

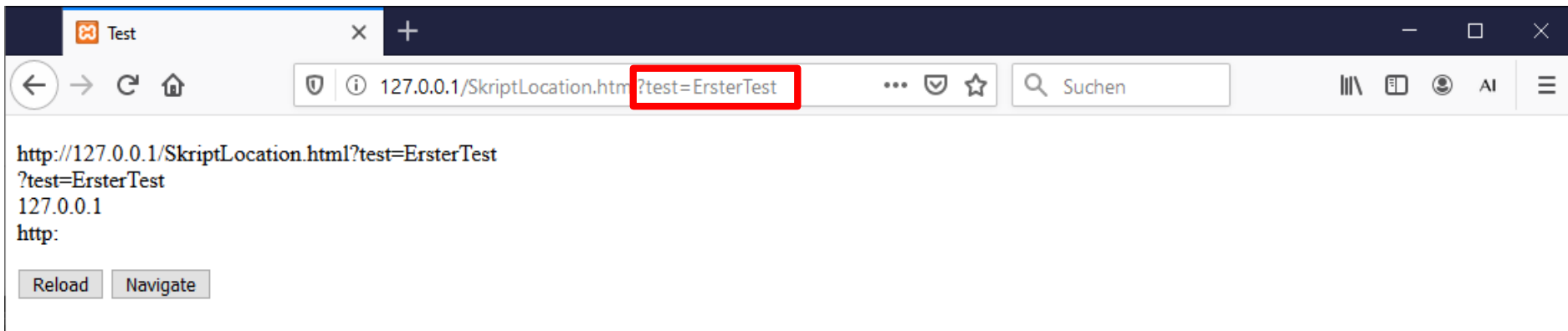
```
<body>
  <form action="foo()">
    <p id="inhalt">Inhalt</p>
    <p><input type="button" value="Reload" onclick="reload()">
      <input type="button" value="Navigate" onclick="navigate()">
    </p>
  </form>

  <script>
    function auslesen() {
      document.getElementById("inhalt").innerHTML= location.href
        + "<br>" + location.search + "<br>" + location.host +
          "<br>" + location.protocol; }
    auslesen();
  //->
```

JavaScript Location

Beispiel

```
function reload() {  
    location.reload();    }  
  
function navigate() {  
    location.href="convert.htm";    }  
    </script>  
</body>
```



JavaScript

Weitere Objekte

History

- `window.document.history`: Zugriff auf die besuchten Seiten
- `history.length`: Anzahl der besuchten Seiten
- `history.back()`: Eine Seite zurück in der History
- `history.forward()`: Eine Seite vor in der History
- `history.go()`: Mehrere Seiten vor oder zurück durch Angabe von positiven oder negativen Werten

JavaScript History

Beispiel

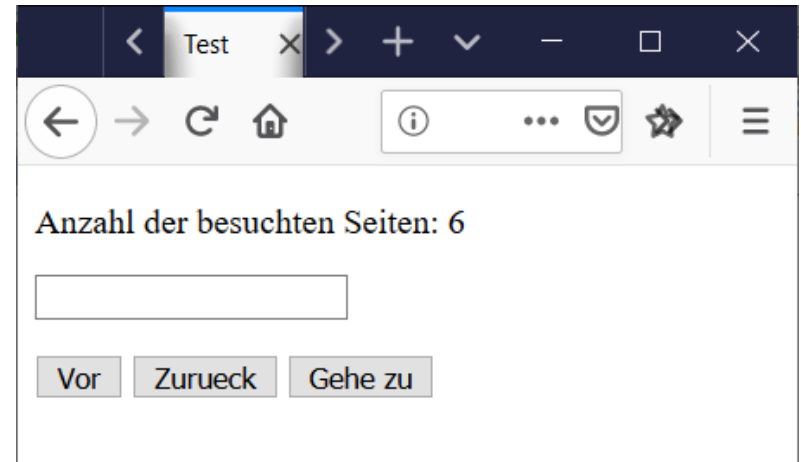
```
<body>
<form action="foo()">
  <p id="inhalt">Azahl der besuchten Seiten: </p>
  <p> <input id="anzahl"></p>
  <p><input type="button" value="Vor" onclick="vor()">
    <input type="button" value="Zurueck" onclick="zurueck()">
    <input type="button" value="Gehe zu" onclick="gehezu()">
  </p>
</form>

<script>
  function auslesen() {
    document.getElementById("inhalt").innerHTML += history.length;
  }
  auslesen();
//-->
```

JavaScript History

Beispiel

```
function vor() {  
    history.forward() ;  
}  
  
function zurueck() {  
    history.back() ;  
}  
  
function gehezu() {  
    history.go(anzahl.value) ;  
}  
  
</script>  
</body>
```



JavaScript

Datenspeicherung

Datenspeicherung im Browser per `cookie` oder `localStorage`

- Daten können vom Browser gespeichert werden.
- Cookies stellen dabei eine Speicherung zur Verfügung, die kleine Datenmengen speichert und teilweise vom HTTP-Protokoll unterstützt wird.
- `localStorage` dient dazu, größere Datenmengen zu speichern. Dies kann beispielsweise dafür benutzt werden, Daten offline zur Verfügung zu stellen.
- `cookie` Speicherung meist bis zu 4 KB Daten
- `localStorage` Speicherung meist bis zu 5 MB Daten

JavaScript

Cookies

- `x = document.cookie` lesen
- `document.cookie= x` schreiben

Folgende Attribute sind vorhanden:

- Name
- Inhalt
- Haltbarkeit
- Pfadname
- Domain
- Secure

5 Tage haltbarer Cookie:

```
var infuenfTagen = ablauf.getTime() + (5 * 24 * 60 * 60 * 1000);
ablauf.setTime(infuenfTagen);
document.cookie=
    "cookieName=cookieValue; expires="+ablauf.toGMTString();
```

JavaScript

Cookies

Beispiel

```
<body>
  <form action="SkriptCookie2.html">
    <p id="cookie">Inhalt des Cookie ausgelesen:</p>
    <input id="cookieName">Name des Cookie</p>
    <input id="cookieContent">Inhalt des Cookie</p>
    <p>
      <input type="button" value="Cookie schreiben"
        onclick="schreiben()">
      <input type="button" value="Cookie lesen" onclick="lesen()">
    </p>
    <p><input type="submit" value="Absenden"></p>
  </form>
//-->
```

JavaScript

Cookies

Beispiel

```
<script>
function lesen() {
    document.getElementById("cookie").innerHTML +=
        (" " + document.cookie);}
function schreiben() {
    let datum = new Date();
    let dauer = 2;
    let cookieName = document.getElementById("cookieName").value;
    let inhalt = document.getElementById("cookieContent").value;
    datum.setTime(datum.getTime() + (dauer*24*60*60*1000));
    let ablaufdatum = "expires=" + datum.toGMTString();
    document.cookie = cookieName + "=" + inhalt + ";" +
        ablaufdatum + "; path=/";
}
</script>
```

JavaScript localStorage

- Schreiben von Name-Wert Paaren in einen lokalen Speicher
- Weitere Funktionen zum Lesen und Entfernen der Inhalte

- `localStorage.setItem(aufgabeNr, aufgabeText);`
- `localStorage.getItem('aufgabeNr');`
- `localStorage.removeItem('aufgabeNr');`
- `localStorage.clear();`

Vgl.: https://wiki.selfhtml.org/wiki/JavaScript/Web_Storage

JavaScript

localStorage

Beispiel

```
<body>
  <form action="SkriptLocalStorage1.html">
    <p id="ls">Inhalt:</p>
    <input id="name">Name des Items</p>
    <input id="wert">Wert des Items</p>
    <p><input type="button" value="Wertepaar schreiben"
      onclick="schreiben()">
    <input type="button" value="Wertepaare lesen"
      onclick="lesen()"> </p>
    <p><input type="button" value="Wertepaar loeschen"
      onclick="loeschen()">
      <input type="button" value="LocalStorage loeschen"
        onclick="alleLoeschen()">
    </p>
    <p><input type="submit" value="Absenden"></p>
  </form>    //-->
```


JavaScript

localStorage

Beispiel

```
<script>
function lesen() {
    let name = document.getElementById("name").value;
    let ausgabe = localStorage.getItem(name) ;
    document.getElementById("ls").innerHTML += ("<br>" + ausgabe);}
function schreiben() {
    let name = document.getElementById("name").value;
    let wert = document.getElementById("wert").value;
    localStorage.setItem(name, wert) ;}
function loeschen() {
    let name = document.getElementById("name").value;
    localStorage.removeItem(name) ;}
function alleLoeschen() {
    localStorage.clear() ;}
</script>
</body>
```