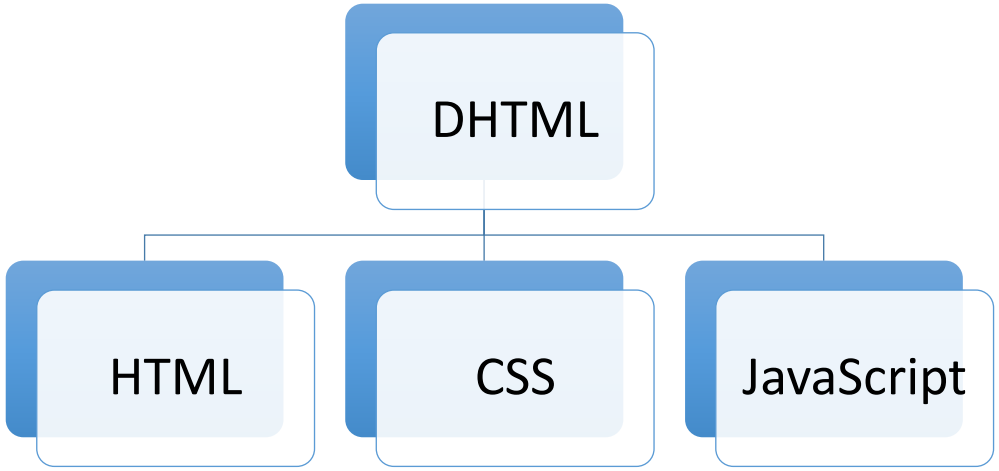
# JavaScript – Grundlagen

Ingo Köster

Diplom Informatiker (FH)

# Was ist JavaScript?

- Interaktive Web-Seiten müssen auf Eingaben eines Anwenders reagieren können
- DHTML (Dynamic HTML) ist notwendig für Seiten, die sich wie Desktop-Anwendungen „anfühlen“ sollen



# Was ist JavaScript? – Teil 2

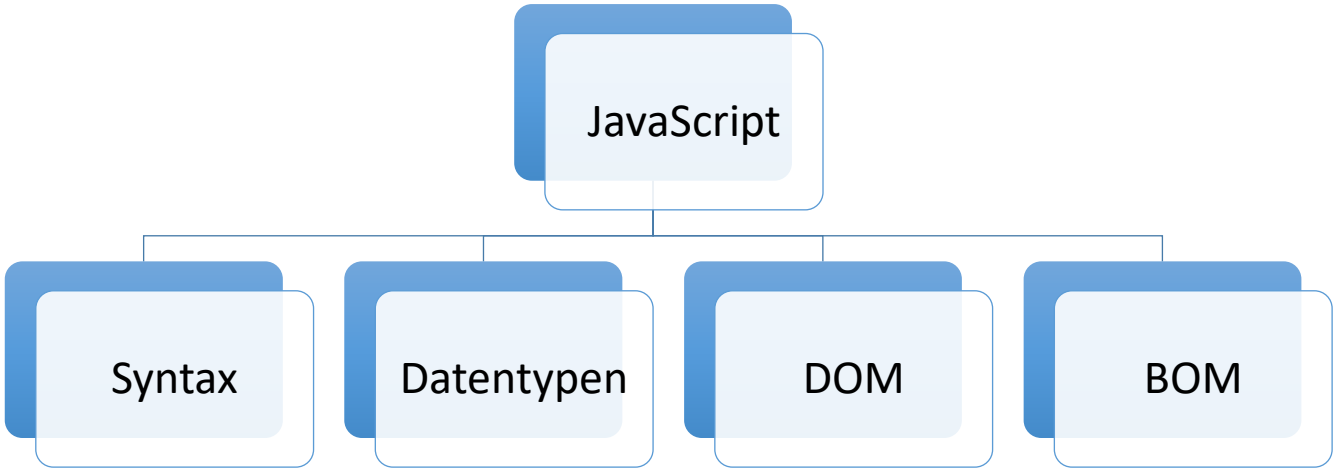
- JavaScript ist eine Entwicklung der Firma Netscape, inzwischen ECMA-Standard
- JavaScript wird durch den Browser ausgeführt (Interpreter/Compiler)
- Sprache mit wenigen Schlüsselworten und Objekten
- Limitierte Möglichkeiten, Browser als Sandbox
- Gut für die Manipulation des DOM
- Reaktion auf Aktionen des Anwenders auf dem Client

# Was ist JavaScript? – Teil 3

› JavaScript ist objektorientiert

› JavaScript ist dynamisch getypt

› Kleiner Sprachumfang, viele interessante Bibliotheken



# Typische Einsatzgebiete

- Eingabevalidierung für Formulare
- Dynamische Anpassungen von Seitenelementen, CSS
- Einfache bis komplexe Berechnungen
- „Neue“ Steuerelemente erzeugen
- AJAX
  - Asynchrone Kommunikation mit einem Web-Server
  - Dynamisch Inhalte ein- und ausblenden
- HTML5 Techniken (Canvas, Web Sockets, Web Worker, etc.)

# Skripte einbinden

- JavaScript wird durch das HTML-Tag `<script>` eingebunden
- Das Tag kann im
  - HEAD und im
  - BODY verwendet werden
- Skripte sollten extern in Dateien gespeichert werden (Dateiendung .js)
- Browser können Skripte cachen

```
<script src="scripts.js">
<!– Code hier wird nicht ausgeführt -->
</script>
```

# Erstes Skript

```
<html>
…
<body>
  <script>
     alert('Hello JavaScript!');
  </script>
</body>

</html>
```

# Syntax – C-ähnlich

- Operatoren (`+, *, =, !=, &&, ++, …`)
- Variablen (dynamische Typen)
- Verzweigungen (`if, else, switch`)
- Schleifen (`for, while`)
- Arrays (indiziert und assoziativ)
- Funktionen
- Funktionen sind Datentypen (ähnelt C#-Delegaten)

# Datentypen – Grundlegendes

- JavaScript kennt fünf „primitive“ Datentypen
  - `Number (inklusive NaN, Infinity, -Infinity)`
  - `String`
  - `Boolean (true, false)`
  - `Undefined`
  - `Null (null)`

- Alle anderen Typen sind Objekte

- Es können Arrays beliebiger Typen gebildet werden

# Besonderheiten

- Wird ein Wert irgendeines Typs in einen Bool-Wert umgewandelt, so ergibt das immer `true`, außer der Wert ist einer der folgenden:
  - `""`
  - `null`
  - `undefined`
  - `0`
  - `NaN`
  - `false`

- Mittels `const` werden Konstanten definiert
  - `const g = 9.80665;`
    - Der Versuch eine Konstante zu ändern, führt nicht zu einer Fehlermeldung! Der Wert bleibt bestehen

# Datentypen – Teil 1

- Number (Reelle Zahlen) / Boolean (`true/false`)
  - `let i = 42; let b = true;`
    - `let zahl = 1_000_000; // numerische Separatoren wie in C#`
- String
  - `let t1 = "Hallo"; let t2 = 'Hallo';`
- `Date` (Objekt)
  - `let heute   = new Date();`
  - `let einTag1 = new Date(2012, 7, 1);`
  - `let einTag2 = new Date(2012, 7, 1, 12, 0, 0);`

# Datentypen – Teil 2

- Alle Zahlen werden in JavaScript als 64Bit Werte gespeichert
- Nach IEEE 754 Standard
- Aus anderen Sprachen als `double` bekannt (z.B. C# oder Java)
- Keine expliziten Integer Variablen (es gibt `BigInt`)
- Wertebereich:
  - -9.007.199.254.740.922 bis 9.007.199.254.740.922
- Ungenauigkeiten bei Fließkommazahlen beachten
  - Bei Währungen wie Euro z.B. mit Cents als Basis rechnen

# Datentypen feststellen

› Mittels `typeof` den Typ von Variablen als `string` ermitteln

| Type | Rückgabestring |
|---|---|
| `Undefined` | `"undefined"` |
| `Null` | `"object"` |
| `Boolean` | `"boolean"` |
| `Zahl` | `"number"` |
| `String` | `"string"` |
| `Funktion` | `"function"` |
| `Alles andere` | `"object"` |

# Datentypen feststellen - Beispiele

```
typeof 42 === 'number';
typeof "" === 'string';
typeof true === 'boolean';
typeof undefined === 'undefined';
typeof {a:42} === 'object';
typeof [1, 2, 3] === 'object';
```

- Mit `Array.isArray` auf Array prüfen

```
typeof Math.sin === 'function';
typeof null === 'object';
```

# Datentypen – Arrays

› Arrays (indiziert)

```
let leer = [];
let stuff = [1, 2, 3.145, "Hallo", "Welt"];
```

› Assoziative Arrays (Hash/Dictionary)

```
let monate = {"Jan": 31, "Feb" : 28};
monate["Mar"] = 31;
monate.Mar    = 31;
```

# Datentypen – Arrays II

- Arrays (indizierte und assoziative) haben in JavaScript viele Funktionen:
  - Sind dynamisch
  - Haben Methoden zum flexiblen Management der Elemente und können so die folgenden Datenstrukturen ersetzen:
    - Liste
    - Stack
    - Queue
  - Assoziative Arrays können Dictionaries und Sets nachbilden

# Datentypen – Arrays III

› In einem Array müssen nicht alle Elemente angegeben werden:

```
let fish = ['Lion', ,'Angel']
fish[1] ist undefined
```

› Ein zusätzliches Komma am Ende der Liste wird ignoriert:

```
let myList = ['home', ,'school',]; // Länge 3
myList[3] existiert nicht
```

› Komma am Anfang:

```
let myList = [ , 'home', , 'school']; // Länge 4
```

› Das letzte Komma wird ignoriert:

```
let myList = ['home', , 'school', , ]; // Länge 4
```

# Datentypen – Teil 3

› Alle „Variablen“ können wie Objekte behandelt werden, d.h. sie haben Instanz-Methoden:

```
let einString = "ein string";
alert(einString[6]);            // das 'r' in einString
alert(einString.charAt(5));     // 't' in einString
alert("test".charAt(1));        // 'e' in test
alert("test".substring(1,3)); // 'es' in test


let arr = [1,3,4];
alert (arr.length);       // 3
arr.push(7);              // hängt 7 an das Ende des Arrays
alert (arr[3]);           // 7
```

# Datentypen – Konvertierung

› Der Operator + ist für Strings überladen:

```
let string1 = "Java";
let string2 = "Script";
alert(string1 + string2);  // JavaScript
```

› Implizite Umwandlungen:

```
alert("9" + 3);  // 93 !!!
alert("9" - 3);  //  6
alert("9" * 3);  // 27
alert("9" / 3);  //  3

alert(3 + "9");  // 39
```

# Datentypen – Konvertierung II

› Texte in Zahlen umwandeln:

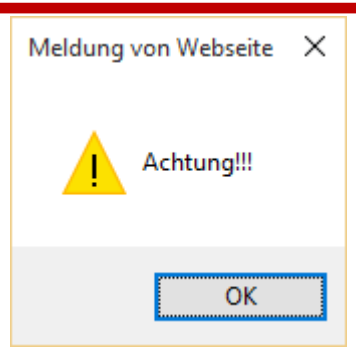
```
let x = parseInt("122");          // ok
let x = parseInt("122", 10);      // besser

let pi = parseFloat("3.145");
```

# Standard-Dialoge

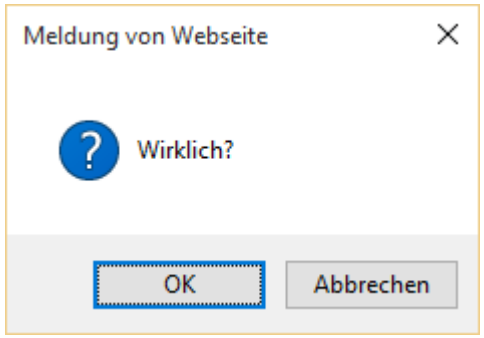
› Hinweise (modal):

```
alert("Achtung!!!");
```



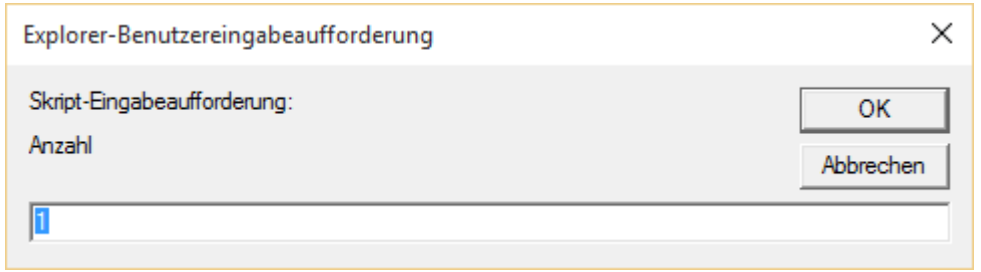

› Bestätigung (modal):

```
confirm("Wirklich?");
```




› Eingabe (modal):

```
prompt("Anzahl", 1);
```



# Verzweigungen – if

| Symbol | Bedeutung |
|---|---|
| > | größer |
| < | kleiner |
| >= | größer oder gleich |
| <= | kleiner oder gleich |
| ==, === | gleich (Wert), gleich (Typ + Wert) |
| != | ungleich |
| !== | nicht gleich und/oder nicht vom selben Datentyp |

```
let unitPrice = 1.30;
if (quantity > 100)
{
  unitPrice = 1.20;
}
```

# Verzweigungen – switch

› Genau wie in C/C++/Java/C#:

```
switch (variable) {
  case 1:
    // Fall 1
    break;
  case 'a':
    // Fall 2
    break;
  case 3.14:
    // Fall 3
    break;
  default:
    // alle anderen Fälle
}
```

# Schleifen

```
let counter;
for (counter = 0; counter < 5; counter++)
{
  alert(counter);
}

counter = 0;
while (counter < 5)
{
  alert(++counter);
}
```

```
counter = 0;
do
{
  alert(++counter);
}
while (counter < 5);
```

# Schleifen - II

› for-Schleifen über Arrays und Objekte:

```
let zahlen = [11, 22, 34, 46, 15];

for (let x in zahlen) {
  alert( zahlen[x] );
}

let obj = { Name : "Thaler", Vorname : "Tim" };

for (let x in obj) {
  alert( obj[x] ); // Liefert Thaler dann Tim
}
```

# Fehlerbehandlung – try/catch

› Wie in C++/Java/C# gibt es `try-catch`

```
let a = {};

try
{
   alert("Ausgabe: " + a.hallo());
   // Fehler: Methode existiert nicht
}
catch(e) {
      console.log(e);
}
Ausgabe:
TypeError {stack: (...), message: "undefined is
not a function"}
```