# JavaScript – Objekte und deren Funktionen

Ingo Köster

Diplom-Informatiker (FH)

# Übersicht

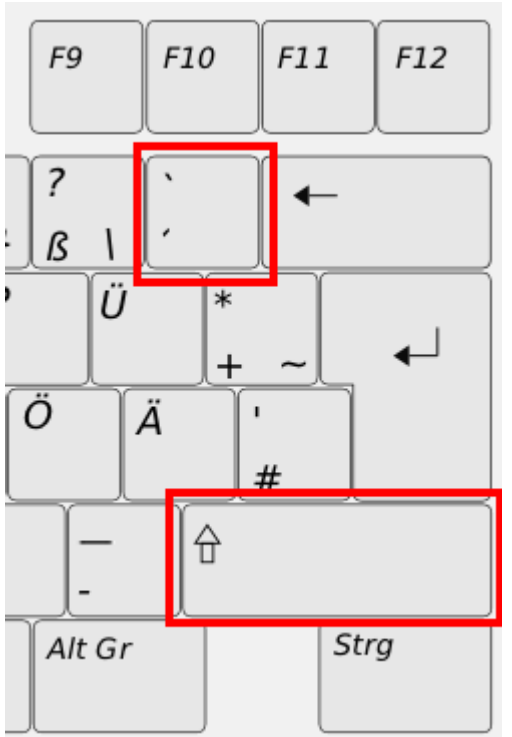
- Zeichenketten
  - String
- Mathematische Berechnungen
  - Math
- Zahlen
  - Number
- Felder
  - Array
- Zeitangaben
  - Date
- Reguläre Ausdrücke
  - RegExp

# String

- Anlegen
  - `let txt1 = "Hallo";`
  - `let txt2 = 'Hallo';`
- Anführungszeichen in einem String zu verwenden, solange sie nicht den Anführungszeichen um den String entsprechen
  - `let txt = "Was'n das?";`
  - `let n = 'Name: "Johnny" ';`
- Alternativ Anführungszeichen mit Backslash (\) “escapen”
  - `let txt = 'Was\'n das?';`
  - `let n = "Name: \"Johnny\" ";`
- Länge eines Strings
  - `let txt = "Hallo Welt";`
  - `let i = txt.length;`

# String Interpolation

- In JavaScript gibt es String Interpolation ähnlich zu C#
- Anstelle von Verkettung von Strings mit + können Variablen in Strings eingesetzt werden
- Diese funktioniert nur mit speziellen Anführungszeichen !!!!
  - Tastenkombination + Leertaste
  - Nicht mit " oder mittels ' !




```
const zahl1 = 18.3;
console.log(`Wert der Zahl ist ${zahl1} `);
```

# String Interpolation

- Operation
  - `const zahl1 = 18.3;`
  - `const zahl2 = 1.7;`
  - `console.log(`Ergebnis: **${zahl1 + zahl2}**`);`

- If-Bedingung
  - `console.log(`**${zahl1 > 10 ? 'AAAA' : 'BBBB'}**`);`

- Funktionsaufruf
  - `console.log(`**${myFunc(100)}**`);`

# In Strings suchen

- Suchen (Erstes Vorkommen)
  - `let str = "Please locate where 'locate' occurs!";`
  - `let pos = str.indexOf("locate");`

- Suchen (Letztes  Vorkommen)
  - `let pos = str.lastIndexOf("locate");`

- Suchen (Reguläre Ausdrücke)
  - `let pos = str.search("locate");`

# String Methoden

- Ersetzen
  - `let str = "Hello Welt";`
  - `let n = str.replace("Welt","World");`
- Zu Großbuchstaben
  - `let txt = "Hello World!";`
  - `let txt1 = txt.toUpperCase();`
- Zu Kleinbuchstaben
  - `let txt1 = txt.toLowerCase();`
- In ein Array umwandeln
  - `let txt = "a,b,c,d,e";`
  - `txt.split(",");`

# Sonderzeichen

| Code | Outputs |
|---|---|
| \' | Einfache Anführungszeichen |
| \" | Doppelte Anführungszeichen |
| \\ | Backslash |
| \n | Neue Zeile |
| \r | Wagenrücklauf |
| \t | Tabulator |
| \b | Backspace |

# Math

| Methode | Beschreibung |
|---|---|
| `abs(x)` | Betrag von x |
| `ceil(x)` | An die nächste ganze Zahl aufrunden |
| `cos(x)` | Cosinus Wert |
| `floor(x)` | An die nächste ganze Zahl abrunden |
| `max(x,y,z,...,n)` | Die größte Zahl aller Parameter |
| `min(x,y,z,...,n)` | Die kleinste Zahl aller Parameter |

# Math

| Methode | Beschreibung |
|---|---|
| `pow(x,y)` | Potenzieren x hoch y |
| `random()` | Zufällige Zahl zwischen 1 und 0 (inklusive 0, exklusive 1) |
| `round(x)` | Kaufmännisch runden |
| `sin(x)` | Sinus von x |
| `sqrt(x)` | Quadratwurzel von x |

Zufallszahl zwischen 0 und 10:

```
Math.floor(Math.random() * 11);
```

# Array

› Arrays (indiziert)

```
let leer = new Array();
// besser
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;
```

# Arrays

- Arrays (indizierte und assoziative) haben in JavaScript viele Funktionen:
  - Sie sind dynamisch
  - Sie haben Methoden zum flexiblen Management der Elemente und können so die folgen Datenstrukturen ersetzten:
    - Liste
    - Stack
    - Queue
  - Assoziative Arrays können Dictionaries und Sets nachbilden

# Arrays anlegen

- 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`

# Array Methoden

- Array als String
  - `let fruits = ["Banana", "Orange", "Apple", "Mango"];`
- Mittels Methoden:
  - `fruits.valueOf();`
  - `fruits.toString();`
  - `fruits.join(" * "); // Separatorzeichen angeben`
- Array sortieren mittels `sort()`
  - Sortiert ein Array immer alphabetisch, auch bei Zahlen! (1,10,100,2,20,…)
  - Sortieren nach Werten z.B. mit Hilfs-Funktion

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

# Array Methoden (Auswahl)

| Methode | Beschreibung |
|---|---|
| `concat()` | 2 oder mehr Arrays zusammenfügen |
| `indexOf()` | Nach einem Element suchen und Position liefern |
| `lastIndexOf()` | Von hinten Suchen und Position liefern |
| `pop()` | Letztes Element entfernen und zurückliefern |
| `push()` | Neues Element am Ende einfügen und neue Länge zurückliefern |
| `reverse()` | Array umdrehen |

Übersicht: https://www.w3schools.com/jsref/jsref_obj_array.asp

# Array Methoden

| Methode | Beschreibung |
|---|---|
| `shift()` | Entfernt das erste Element und liefert es zurück |
| `slice()` | Teil eines Arrays auswählen |
| `splice()` | Hinzufügen bzw. Entfernen von Elementen |
| `unshift()` | Element am Anfang einfügen und die neue Länge zurückliefern |

```
let fruits = ["Banana","Orange","Lemon","Apple","Mango"];
let citrus = fruits.slice(1, 3); // "Orange", "Lemon"

let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2,0,"Lemon","Kiwi"); // index, wie viele,
item1, item2, …
```

Ergebnis: Banana,Orange,Lemon,Kiwi,Apple,Mango

# Array Methoden

- Die Funktion `map()` wendet auf jedes Element eines Arrays die gleiche Funktion an
  - Die Funktion liefert ein neues Array als Ergebnis zurück

- Beispiel
  - `let zahlen = [1, 4, 9];`
  - `let wurzel = zahlen.map(Math.sqrt);  // 1,2,3`
    - Array zahlen bleibt 1,4,9

- Andere Funktion:
  - `let d = zahlen.map(n => n * 2);`

# Mehrdimensionale Arrays

```
let elemente = [
  [1, 2],
  [3, 4],
  [5, 6]
];
console.log(elemente[0][0]);      // 1
console.log(elemente[0]);         // [1, 2]
```

# RegExp

- Syntax
  - `/pattern/modifiers;`
  - /[a-d]/i  Ist ein regulärer Ausdruck in JavaScript
  - [a-d]  Ist das Suchmuster
  - i  Ist ein Modifizierer

- Modifizierer
  - i  Groß- bzw. Kleinschreibung ignorieren
  - g  Globale Suche (ohne g wird nach dem ersten Treffer aufgehört)
  - m  Über mehrere Zeilen suchen

# Reguläre Ausdrücke

- Klammern
  - [abc] Eines der Zeichen in den Klammern
  - [^abc] Keines der Zeichen in den Klammern
  - [0-9] Ziffer zwischen 0 und 9
  - (x|y) Einer der beiden Ausdrücke (getrennt durch das |)

- Spezielle Ausdrücke
  - . Beliebiges Zeichen
  - \d Eine Ziffer
  - \s Ein Leerzeichen
  - \b Wortanfang bzw. Wortende

# Reguläre Ausdrücke

- Quantifzierer
  - n+ Muster kommt mindestens 1 oder mehrfach vor
  - n* Muster kommt gar nicht, einmal oder mehrfach vor
  - n? Muster kommt gar nicht oder genau einmal vor

- Beispiel

```
/<a.*>/g
```

- Zeichenfolgen, die mit „<a“ beginnen, gefolgt von beliebig vielen Zeichen und mit „>“ enden

# Reguläre Ausdrücke `test()`

- Liefert true oder false
  - `let str = "Hallo Welt!";`
  - `let pattern = new RegExp("e");`
  - `let result = pattern.test(str);  // true`

  - `let str = "Hallo1 Welt!";`
  - `let pattern = new RegExp("\\d");`
  - `let result = pattern.test(str);  // true`

# Number

- Konstruktor
  - `new Number(wert);`

- Beispiele:
  - `let zahl = new Number(42);`
  - `let x = new Number("23");`

- Ist der Übergabeparameter keine Zahl wird NaN zurückgeliefert

# Number Konstanten

- MAX_VALUE
  - Gibt die größte darstellbare Zahl wieder
  - `let x = Number.MAX_VALUE; // ~ 1.79E+308`

- MIN_VALUE
  - Kleinste positive Zahl, die am nächsten an 0 (Null) liegt
  - `let x = Number.MIN_VALUE; // ~ 5E-324`

# Number Methoden

- `Number.isNaN(wert)`
  - Überprüfen ob wert NaN ist

- `Number.isInteger(wert)`
  - Überprüfen ob wert eine Zahl ist (`Number`) und ein Ganzzahl-Wert ist
  - `Number.isInteger(3)    // true`
  - `Number.isInteger(3.14) // false`

# Number Methoden

- `Number.parseFloat()`
  - Wie `parseFloat` des globalen Objektes `window`
  - `parseFloat(string, radix);`

- `Number.parseInt()`
  - Wie `parseInt` des globalen Objektes `window`
  - `parseInt(string, radix);`

# Datentypen – Konvertierung

› Texte in Zahlen umwandeln:

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

› let pi = parseFloat("3.145");
```

# Date

› Datumsangaben berechnen

```
let myDate = new Date();
myDate.setDate(myDate.getDate() + 5);
```

› Datumsangaben vergleichen

```
let x = new Date(2152,2,2);
let today = new Date();
if (x > today)
{ … }
else
{ … }
```

# debugger

› Funktion `debugger` verhält sich beim Aufruf wie ein Breakpoint

```
function f(o)
{
  if (o === undefined)
  {
    debugger;
  }
}
```

› Wird `f()` ohne Übergabeparameter aufgerufen, hält die Verarbeitung des Skriptes zum Debuggen an