INP (07)

Prof. Dr.-Ing. S. Gössner

University of Applied Sciences Lippe & Höxter

Inhalt

Datentyp Array

Javascript besitzt neben einfachen Datentypen wie bspw. `Number` und `Boolean` auch komplexe Datentypen.

`Array` ist solch ein komplexer Datentyp.

- Einem einfachen Datentyp kann genau **ein Wert** zugeordnet werden, dem Datentyp `Array` dagegen **eine Menge von Werten**.
- Arrays werden verwendet, um logisch zusammenhängende Werte zu gruppieren.
- Ein Array kann eine beliebige, nicht ausdrücklich begrenzte Anzahl von Werten aufnehmen.
- Im Gegensatz zu streng typisierten Programmiersprachen müssen in Javascript die Datentypen der einzelnen Arraywerte **nicht gleich** sein.
- Der Zugriff auf die einzelnen Werte des Arrays erfolgt über einen *Index*. Im Sinne der Eindeutigkeit spricht man deshalb auch von **indizierten Arrays**.
- Javascript Arrays sind dynamisch und müssen bei der Definition hinsichtlich ihrer Grösse nicht festgelegt werden.

Array Literale

Javascript unterstützt *Array Literale*. Dies ist eine einfache Schreibweise zur Definition von Array-Werten.

Syntax

```
[wert1, wert2, wert3, ... wertN]
```

Das Array Literal ist eine durch *Kommata* getrennte Liste von Werten, die von *eckigen Klammern* eingeschlossen ist.

Wie die meisten Literale in Javascript wird auch das Array Literal zur Initialisierung von Variablen benutzt.

```
var arr = ["Jim","Bob"];
```

Fehlende Array Elemente werden durch mehrere aufeinanderfolgende Kommata gekennzeichnet. Der Wert fehlender Array Elemente is `undefined`.

```
var arr = ["Jim","Bob",,"Joe"];
```

Die Anzahl der Elemente eines Arrays erhalten wir über die *Eigenschaft* `length`. Fehlende Elemente werden mitgezählt.

```
var arr = ["Jim","Bob",,"Joe"];
var len = arr.length;  // len = 4
```

Ein Javascript Array kann gemischte Datentypen beinhalten.

```
var arr = ["one", true, "three", 4];
```

Häufig wird zu Beginn ein leeres Array benötigt.

```
var arr = [];  // empty array
```

Array Elemente können Werte von *Ausdrücken* zugewiesen bekommen.

```
var x = 3, arr = [x, 2*x-1, 3*x-1];
```

Array Konstruktor

Die Array Initialisierung mittels des *Array Konstruktors* ist die ältere Variante.

Syntax

Leeres Array

```
new Array()
```

Array mit *n* Elementen vom Typ `undefined`

```
new Array(n)
```

Array mit den Elementen `wert1, wert2 ... wertN`

```
new Array(wert1, wert2 ... wertN)
```

Beispiel

```
var arr = new Array("Jim","Bob","Joe");
```

Heute werden Arrays meist bequem mittels Literale initialisiert.

Array Operator

Der Zugriff auf einzelne Array Elemente erfolgt mittels des *Array Operators* `[]`.

Syntax

```
arrayName[index]
```

Die Indizierung der Elemente eines Array beginnt grundsätzlich bei 0. Der Index des letzten Elements ist also immer `length-1`.

```
var arr = ["Jim","Bob",,"Joe"];
write(arr[0]); // Jim
write(arr[2]); // undefined
write(arr[3]); // Joe
```

Der Zugriff auf Array Elemente kann *lesend* und *schreibend* erfolgen.

```
var arr = ["Jim","Bob",,"Joe"];
arr[2] = arr[0] + "-" + arr[1]; // Jim-Bob
```

Die einfachste Möglichkeit, einem existierenden Array ein weiteres Element hinzuzufügen, ist die Zuweisung über den Index `length`.

```
var arr = ["Jim","Bob",,"Joe"];
arr[arr.length] = "Max"; // ["Jim","Bob",,"Joe","Max"]
```

Es können beliebige, gegenwärtig nicht definierte Array Elemente belegt werden.

```
var arr = ["Jim","Bob",,"Joe"];
arr[10] = "Max"; // ["Jim","Bob",,"Joe",,,,,,,"Max"]
print(arr.length); // 11
```

Der übliche Zugriff auf alle Array Elemente der Reihe nach erfolgt mittels der `for`-Anweisung.

```
var arr = ["Jim","Bob",,"Joe"];
for (var idx=0; idx<arr.length; idx++)
  print(arr[idx]); // 11
```

Mehrdimensionale Arrays

Javascript unterstützt nicht ausdrücklich *mehrdimensionale Arrays*. Es ist jedoch möglich, Arrays als Array Elemente zu definieren.

Syntax

```
[[wert11,...,wert1N], [wert21,...,wert2N], ... ,[wertM1,.
```

Der Zugriff auf solche inneren Array Elemente erfolgt über aneinandergehängte Array Operatoren.

```
var arr = [[1,2], [3,4], [5,6]];
print(arr[1][0]); // 3
```

Die inneren Arrays müssen nicht alle dieselbe Länge besitzen.

```
var arr = [[1,2], [3,4,5], [6,7,8,9]];
print(arr[1].length); // 3
```

Die Bearbeitung aller Elemente eines mehrdimensionalen Arrays erfolgt vorzugsweise mittels verschachtelter Schleifen.

```
var arr = [[1,2], [3,4,5], [6,7,8,9]];
for (var i=0; i<arr.length; i++)
   for (var j=0; j<arr[i].length; j++)
      print(arr[i][j]);
```

Array Methoden

Der Datentyp *Array* stellt eine Reihe nützlicher Methoden zur Verfügung.

Syntax

```
arrayName.methode(arg1, ... , argN);
```

Einige Methoden verändern das Array, einige tun das nicht.

`concat(wert1,...,wertN)`

Verknüpfung eines existierenden Arrays mit weiteren Werten durch Anhängen. Das betreffende Array wird **nicht** verändert.

```
var a = [1,2,3], b = ['A','B','C'], c;
c = a.concat(4,b); // c = [1,2,3,4,'A','B','C']
```

`join(trenner)`

Wandeln aller Array Elemente in Strings und verbinden aller Elemente mittels des gegeben *Trenners* untereinander. Das betreffende Array wird **nicht** verändert.

```
var arr = [0,8,15], s;
s = arr.join("-"); // s = "0-8-15"
```

`pop()`

Entfernen des letzten Array Elements.

```
var arr = ["Jim","Bob","Joe"];
arr.pop(); // arr = ["Jim","Bob"]
```

`push(wert1,...,wertN)`

Anfügen weiterer Werte an ein Array.

```
var arr = [1,2,3];
arr.push(4,5); // arr = [1,2,3,4,5]
```

`reverse()`

Umkehrung der Reihenfolge aller Werte eines Arrays.

```
var arr = [1,2,3];
arr.reverse(); // arr = [3,2,1]
```

shift()

Entfernen des ersten Array Elements.

```
var arr = ["Jim","Bob","Joe"];
arr.shift(); // arr = ["Bob","Joe"]
```

sort()

Lexikalische Sortierung eines Arrays.

```
var arr = ["Jim","Bob","Joe"];
arr.sort(); // arr = ["Bob","Jim","Joe"]
```

unshift(wert1,...,wertN)

Einfügen weiterer Werte an den Anfang eines Arrays.

```
var arr = [1,2,3];
arr.unshift(-1,0); // arr = [-1,0,1,2,3]
```

Beispiel

Benutzereingabe mehrerer Namen (Abschliessen mittels leerer Eingabe) und Ausgabe der lexikalisch sortierten Namen.

```
var names = [], name;
do {
   name = window.prompt("Name bitte!", "");
   if (name != "")
      names[names.length] = name; // or .. names.push(nam
}
while (name != "")
names.sort();
window.alert(names);
```

Datentyp Object

`Object` ist neben `Array` ein weiterer komplexer Datentyp in Javascript.

- Wie ein Array kann auch ein Object mehrere Werte enthalten.
- Im Gegensatz zum Array wird jedem Wert ein Name vom Datentyp *String* zugeordnet.
- Der Zugriff auf einen Wert erfolgt über den zugeordneten Namen.
- Die Werte innerhalb eines Objekts werden als *Eigenschaften* bezeichnet.
- Der Zugriff auf die Eigenschaften erfolgt über die zugeordneten Namen. Wegen ihrer Ähnlichkeit zu Arrays werden Javascript Objects auch als **assoziative Arrays** bezeichnet.
- Objects sind dynamisch. Auch nach ihrer Initialisierung können ihnen Eigenschaften hinzugefügt werden.

Object Literale

Ein `Object` ist eine Kollektion von *Name/Wert*-Paaren.

Syntax

```
{ name1: wert1, name2: wert2, ... , nameN: wertN }
```

Ein Object Literal wird vorwiegend zur Initialisierung von Variablen verwendet.

```
var student = {
  name: "Müller",
  matNr: 987654321
};
```

Der Name einer Eigenschaft kann bzw. muss gelegentlich in Anführungszeichen geschrieben werden.

```
var duration = {
  "in": "15.10",
  out: "16.20"
};
```

Ein leeres Objekt kann einfach definiert werden.

```
var obj = {};
```

Bevor die literale Objektnotation in Javascript übernommen wurde, konnten Objekte lediglich mittels

```
var obj = new Object();
```

definiert werden. In vielen Beispielen, Büchern, Tutorials ist daher noch diese veraltete – jedoch nach wie vor unterstützte – Form zu finden.

Objekt Bearbeitung

Auf die Eigenschaften eines Objekts kann sowohl mittels der sogenannten *dot*-Notation, als auch mittels der *Array*-Notation lesend und schreibend zugegriffen werden.

Syntax

```
obj.eigenschaft;       // dot-Notation
obj["eigenschaft"];    // array-Notation
```

Die Eigenschaften eines Objekts können verändert werden.

```
var punkt = {
  x: 25.3,
  y: -6.7
}
punkt.y = 0;
```

Einem Objekt können nachträglich Eigenschaften zugewiesen werden.

```
var punkt = {
  x: 25.3,
  y: -6.7
}
punkt.z = 12;
```

Objekte lassen sich verschachteln.

```
var linie = {
  p1: { x: 25.3, y: -6.7 },
  p2: { x: 37.2, y: 12.9 }
};
var dx = linie.p2.x - linie.p1.x;
```

Objekte können Arrays beinhalten.

```
var polyline = {
  color: "red",
  xcoords: [25.3, 37.2, 29.7],
  ycoords: [-6.7, 12.9, 2.3]
};
var p2 = {
  x: polylinie.xcoords[1],
  y: polylinie.ycoords[1]
};
```

Arrays können Objekte enthalten

```
var points = [
  { x: 25.3, y: -6.7 },
  { x: 37.2, y: 12.9 },
  { x: 29.7, y:  2.3 }
];
```

Die Bearbeitung aller Eigenschaften eines Objekts kann mittels der `for .. in` – Schleife erfolgen.

```
var student = [
  name: "Müller",
  vorname: "Heinz",
  matnr: 987654321
};

for (eigenschaft in student)
   print(eigenschaft + "=" + student[eigenschaft]);
```