**Arrays**

Array = Liste von Elementen bzw. geordnete Sammlung von Daten   
werden verwendet, um mehrere Werte in einziger Variablen zu speichern (Nicht-Array-Variable kann nur einen einzigen Wert speichern)

lassen sich erzeugen mithilfe von eckigen Klammern []

let hunde = new Array(); // erzeugt leeres Array

let hunde = new Array("Pinscher","Dackel","Hund","Boxer"); // erzeugt Array mit Werten

**Kurzschreibweise:**

let colors = []; // erzeugt leeres Array

let meinArray = [1, 2, 3, 4];

**Anzahl der Elemente im Array bestimmen:**

console.log(meinArray.length); // => 4

**Array-Zugriff mit Indexoperator**

jedes Element in Array hat Nummer - auch numerischer Index genannt, über die man auf Element zugreifen kann (in JS beginnen Arrays mit Index 0)

Elemente innerhalb des Arrays stehen in Reihenfolge – wie nummerierte Liste  
können anhand ihrer Position auf einzelne Elemente zugreifen

0 - Element 1  
1 - Element 2  
2 - Element 3  
3 - Element 4  
4 - Element 5

Nummerierung = Index und Namen = Werte des Arrays

Zugriff auf einzelnen Namen mit Indexoperator - die eckigen Klammern []   
werden direkt hinter Array notiert  
innerhalb der Klammern wird Index des gesuchten Elements angegeben

console.log(meinArray[2]); // => 3

**Array-Manipulation**

**push() – zum hinzufügen weiterer Elemente**

Funktion push() für Aufnahme weiterer Elemente ins Array - Element anhängen

neues Element wird der Funktion push als Argument übergeben  
im Array taucht es an letzter Position auf

**pop() – zum entfernen von Elementen**

pop() entfernt Element am Ende des Arrays  
Funktion gibt entferntes Element zurück, so dass es weiterverwendet werden kann

**unshift()**

unshift() Methode fügt ein oder mehrere Elemente am Anfang eines Array hinzu   
gibt neue Länge des Arrays zurück

**shift()**

Methode shift() entfernt erstes Element eines Arrays und gibt Element zurück  
shift() verändert Länge des Arrays - ist length = 0, wird undefined zurückgegeben

**manipulieren mit splice()**

pop() entfernt Sie immer nur letztes Element eines Arrays  
splice() erlaubt, Elemente an beliebiger Position zu löschen

erstes Argument enthält Position ab der Elemente entfernt werden sollen  
zweites Argument bestimmt Anzahl der zu löschenden Elemente (0-…)

weitere Argumente sind neu hinzuzufügende Elemente   
kann an Index-Position gleichzeitig Löschen und Hinzufügen

**Array-Methoden**

**sortieren mit sort()**

in Grundvariante:  
alle Elemente werden in Strings umgewandelt  
sortiert Array anhand seines UTF-16-Codepoints – d.h. Zahlen werden auch als Strings verglichen  
 = lexiographischer Vergleich der Elemente anhand der ASCII-Tabelle

im Aufbaukurs werden wir eigene Callback-Function schreiben, um hier einzugreifen

**join** **wandelt in Strings**

Funktion join() kann Array in String verwandeln

verbindet hierzu einzelne Elemente mit Separator (auch Zeilenumbruch \n als Separator möglich, um Listenausgabe, bei der Elemente untereinanderstehen, zu erhalten)

**slice() erstellt Kopien**

slice() Methode schreibt Kopie von Teil des Arrays in neues Array-Objekt von begin bis end (end nicht enthalten) - originales Array wird nicht verändert  
bei negativen Index kennzeichnet end Versatz vom Ende der Sequenz: slice(2, -1) extrahiert vom dritten bis vorletzten Element der Sequenz

**indexOf() findet Position von Elementen**

Funktion indexOf() - Einsatz um Position von Element zu finden  
bei keinem Fund, gibt sie -1 zurück

**Strings - Arrays - Strings / split() und join()**

split() zerlegt String anhand eines Separators in Array   
join() konvertiert Array zu String – Seperator kann übergeben werden

**concat() führt Arrays zusammen**

Methode concat() führt zwei oder mehr Arrays zu einem zusammen  
ändert keine existierenden Arrays, gibt stattdessen neues zurück

kann auch Array und neue Arraywerte zu neuem Array zusammenführen

**includes() prüft Array**

prüft, ob Element im Array vorkommt (ab ECMAScript 2016)  
kann Startparameter als zweites Argument übergeben

**Referenz**   
let a = ["a", "b", "c", "d"];

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Funktion** | **Zweck** | **Beispielaufruf** | **a.join("")** | **Rückgabewert**  **(Datentyp)** |
| push | fügt Elemente am Ende des Arrays an | a.push("x", "y") | abcdxy | 6 (number) |
| pop | entfernt das Element am Ende des Arrays | a.pop() | abc | d (string) |
| unshift | fügt Elemente am Anfang des Arrays hinzu | a.unshift("x","y") | xyabcd | 6 (number) |
| shift | entfernt das Element am Anfang des Arrays | a.shift() | bcd | a (string) |
| slice | fertigt eine Kopie von einem Teil des Arrays an | a.slice(2, 4) | abcd | c,d (array) |
| splice | entfernt und/oder ergänzt Elemente im Array | a.splice(2, 2, "x") | abx | c,d (array) |
| sort | sortiert das Array | a.sort() | abcd | a,b,c,d (array) |
| reverse | kehrt die Reihenfolge der Elemente um | a.reverse() | dcba | d,c,b,a (array) |
| concat | Verbindet ein Array mit einem oder mehreren Elementen oder Arrays | a.concat(["x","y"]) | abcd | a,b,c,d,x,y (array) |
| includes (\*) | Prüft, ob ein Element im Array vorkommt | a.includes("x"[,startindex]) | abcd | false (boolean) |
| indexOf | Ermittelt die Position (index) eines Elements im Array | a.indexOf("c"[,startindex]) | abcd | 2 (number) |
| lastIndexOf | Ermittelt die Position (index) eines Elements im Array (vom Ende her) | a.lastIndexOf("c"[,startindex]) | abcd | 2 (number) |
| join | Verbindet alle Elemente eines Arrays zu einem String | a.join("==") | abcd | a==b==c==d (string) |
| toString | wie join, verwendet aber das Komma als festen Separator | a.toString() | abcd | a,b,c,d (string) |

\* ab ECMAScript 2016

**Objekte**

**vordefinierte, browserunabhängige Objekte**

Objekte besitzen:

* Eigenschaften (objektgebundene Variablen)
* Methoden (objektgebundene Funktionen)

Eigenschaften und Methoden werden mit einem Punkt an das Objekt angehängt.

Math.PI

Math.sin()

browserunabhängigen Objekte von JavaScript:

* Object
* Array
* Boolean
* Date
* Function
* Math
* Number
* RegExp
* String

**Object**

* oberstes Objekt in JavaScript
* alle anderen Objekte stammen von ihm ab
* zum Erzeugen eigener Objekte

**Objekte erstellen**

Objektdefinition

* + Konstruktor
  + Hinzufügen

**Objekt in Literal-Schreibweise erstellen**

let myObject = {

// eigenschaft: wert,

// eigenschaft: wert,

// methode: function() { alert(this.eigenschaft); }

};

console.log(myObject.key);

myObject. methode();

Namen mit Leerzeichen, Bindestrich oder reservierten Wörtern in "" oder ''

**Schlüsselwort this** bezieht sich immer auf das entsprechende Objekt

**Leere Objekte**

können Objekte wie Arrays auch komplett leer anlegen   
Attribute werden später schrittweise ergänzt

let myObject = {};

myObject.eigenschaft = "Wert";  
myObject['name der eigenschaft'] = 'Wert';

**Eigenschaft löschen:**delete myObject.eigenschaft

**Objekt mit new erstellen**

let myObject = new Object();

myObject.eigenschaft = 'Wert'

myObject.methode = function(){

alert(this.eigenschaft);

};

Objektschreibweise erlaubt, Eigenschaften zu gruppieren, die zusammengehörig ein Objekt beschreiben

innerhalb der geschweiften Klammern {} werden Eigenschaften (Properties) gruppiert  
hierzu werden sogenannte Key/Value-Pairs (dt: Name/Wert-Paare) genutzt

Name der Eigenschaft (key): Wert der Eigenschaft (value)

Key/Value-Pairs innerhalb des Objektes durch Komma getrennt   
nach letztem Key/Value-Pair ist Komma optional

{  
key1: value1,  
key2: value2,  
...  
}

kann von außerhalb eines Objektes auf seine Property zugreifen:  
myObject.eigenschaft // "Wert"

mit console.dir(myObject) tabellarische Auflistung der einzelnen Properties und ihrer Werte

Änderung von Eigenschaften durch Zuweisung:  
myObject.eigenschaft = "Neuer Wert";

statt Punktschreibweise auch Index-Schreibweise möglich: myObject['name der eigenschaft']

auch Zuweisung funktioniert mit Indexoperator: myObject['name der eigenschaft'] = "Neuer Wert";

ist sinnvoller, kürzere Punkt-Schreibweise zu verwenden   
nur falls Key noch nicht bekannt und aus Variable kommt, ist es nötig, Klammernschreibweise zu verwenden - z. B.:  
const fieldToZeroOut = "price";  
product[fieldToZeroOut] = 0;

im Beispielcode soll Feld auf Wert 0 gesetzt werden

welches Feld das ist, ergibt sich aber erst bei Programmausführung  
Wert von fieldToZeroOut könnte aus Usereingabe stammen

**Hinzufügen neuer Properties**

Properties in JavaScript voll dynamisch  
jederzeit möglich, zu bestehenden Objekten neue Properties hinzufügen, oder bestehende wieder zu entfernen

Objekt sortiert neue Eigenschaft nicht an spezieller Stelle ein  
einzelne Eigenschaftsnamen (keys) haben untereinander keine Reihenfolge

bei Ausgabe sortiert console.log bzw. console.dir sie üblicherweise alphabetisch

typische Funktion in Tiefen des Shops könnte so aussehen:  
let cartProduct = (customerLastname, customerFirstname, customerAddress, productName, productPrice, productCategory, availableSince, numberInStock) => {  
 // Statements  
};

Funktionen mit derart vielen Parametern in Praxis unhandlich   
führen auch zu schwer lesbarem Code und sind fehleranfällig (falsche Reihenfolge oder Anzahl der Argumente…)

**gilt als schlechte Praxis, Funktionen mit mehr als drei Argumenten zu befüttern!**

Objekte ermöglichen, zusammengehörige Werte zu gruppieren und an Funktion zu übergeben