



CODERS.BAY

JAVAScript TEIL 2

FUNKTIONEN, ARRAYS UND OBJEKTE



Man kann mehrere Codezeilen in einem Container zusammenfassen und ihnen einen Namen geben. Dieser wird dann ***Funktion*** genannt. Man kann aber auch einen Haufen Variablen nehmen, sie in einen Container packen und diesen mit einem Namen versehen. In diesem Fall wird dieser dann ***Array*** genannt.



FUNKTIONEN ERKLÄRT ANHAND EINES LIEDES

- Ein Lied hat grundlegend immer zwei Hauptbestandteile. Einen *Refrain* und mehrere *Strophen*. Aufgebaut meist in:
Strophe 1
Refrain
Strophe 2
Refrain
usw...
- So etwas wird **Ablaufplan** genannt. Im Ablaufplan kümmert man sich nicht um die Einzelheiten einer bestimmten Strophe oder eines einzelnen Refrains. Der Ablaufplan spiegelt das große Ganze wider, und man kann sich dann jede Strophe und jeden Refrain vornehmen, um die Einzelheiten zu erfahren. Die Strophen und Refrains in dem Beispiel stehen für Funktionen.

```
<script>
  function refrain1() {
    var text = "... and they all go marching down \n";
    text += "to the ground \n";
    text += "to get out \n";
    text += "of the rain. \n";
    text += " \n";
    text += "boom boom boom boom boom boom boom boom \n";
    alert(text);
  } // Ende Refrain

  function strophe1() {
    var text = "The ants go marching 1 by 1 hurrah, hurrah \n";
    text += "The ants go marching 1 by 1 hurrah, hurrah \n";
    text += "The ants go marching 1 by 1 \n";
    text += "The little one stops to suck his thumb \n";
    alert(text);
  } //Ende Strophe 1

  function strophe2() {
    var text = "The ants go marching 2 by 2 hurrah, hurrah \n";
    text += "The ants go marching 2 by 2 hurrah, hurrah \n";
    text += "The ants go marching 2 by 2 \n";
    text += "The little one stops to tie his shoe \n";
    alert(text);
  } //Ende Strophe 2

  // Hauptprogramm
  strophe1();
  refrain();
  strophe2();
  refrain();
</script>
```



FUNKTIONEN ERKLÄRT ANHAND EINES LIEDES

Der Programmcode gruppiert die Teile des Liedes auf die gleiche Weise, wie es bei einem auf Papier gedruckten Liedtext der Fall ist.

- 💡 **Es wurde eine Funktion angelegt, die `refrain()` heißt.** Bei Funktionen handelt es sich um Codezeilen, die als Einheit unter einem Namen zusammengefasst werden.
- 💡 **Der gesamte Text des Refrains landet in dieser Funktion.** Alles, was als Teil des Refrains ausgegeben wird, steht in der Funktion `refrain()`.
- 💡 **Auch zu jeder Strophe gibt es eine Funktion.** Auch aus dem Text der einzelnen Strophen wurde jeweils eine Funktion gemacht
- 💡 **Der zentrale Code hat die Form eines Ablaufplans.** Nachdem die Einzelheiten an Funktionen übergeben worden sind, steuert das recht kompakte Hauptprogramm nur noch die Reihenfolge, in der die Funktionen aufgerufen werden.
- 💡 **Einzelheiten werden in die Funktionen ausgelagert.** Das Hauptprogramm kümmert sich um das große Ganze. Die Einzelheiten (wie der Refrain und die Strophen ausgegeben werden) werden in den Funktionen festgelegt.

```
<script>
  function refrain1() {
    var text = "... and they all go marching down \n";
    text += "to the ground \n";
    text += "to get out \n";
    text += "of the rain. \n";
    text += " \n";
    text += "boom boom boom boom boom boom boom boom \n";
    alert(text);
  } // Ende Refrain

  function strophe1() {
    var text = "The ants go marching 1 by 1 hurrah, hurrah \n";
    text += "The ants go marching 1 by 1 hurrah, hurrah \n";
    text += "The ants go marching 1 by 1 \n";
    text += "The little one stops to suck his thumb \n";
    alert(text);
  } //Ende Strophe 1

  function strophe2() {
    var text = "The ants go marching 2 by 2 hurrah, hurrah \n";
    text += "The ants go marching 2 by 2 hurrah, hurrah \n";
    text += "The ants go marching 2 by 2 \n";
    text += "The little one stops to tie his shoe \n";
    alert(text);
  } //Ende Strophe 2

  // Hauptprogramm
  strophe1();
  refrain();
  strophe2();
  refrain();
</script>
```

FUNKTIONEN

- 💡 Funktionen sind ausgesprochen nützliche Werkzeuge, wenn es darum geht, in komplexen Programmen die Übersicht zu bewahren. Sie können ein großes, kompliziertes Programm nehmen und es in kleinere Stücke aufbrechen oder modularisieren. Jedes dieser Module steht für sich und löst ein bestimmtes Teil des Gesamtproblems.
- 💡 Stell dir eine Funktion als ein Programm im Miniaturformat vor. Du kannst in Funktionen Variablen definieren, Schleifen und Verzweigungen einbauen und alles machen, was du mit einem Programm machen würdest. Ein Programm, das Funktionen verwendet, ist eigentlich nichts anderes als ein Programm mit Unterprogrammen, die auch Subroutinen genannt werden.

DATEN AN FUNKTIONEN ÜBERGEBEN UND VON IHNEN ÜBERNEHMEN

- Der Code von Funktionen wird logisch vom Hauptprogramm getrennt.
- **Diese Funktionen geben einen Wert zurück.** Sie rufen nicht länger eine eigene **alert**-Anweisung auf. Sie erzeugen stattdessen einen Wert, den sie an das Hauptprogramm übergeben
- **Es gibt nur eine Strophenfunktion.** Da sich die Strophen ziemlich ähneln, ist es sinnvoll nur eine Strophenfunktion zu haben. Diese verbesserte Funktion muss wissen, welche Strophe gerade abgearbeitet wird, um mit den Unterschieden umgehen zu können.
- In diesem Beispiel übernehmen nicht die Funktionen die Ausgabe. Stattdessen sammeln sie Informationen, die sie an das Hauptprogramm zurückgeben. Im Hauptprogramm wird jede Funktion wie eine Variabel behandelt. Dadurch werden Programme flexibler.
- Die Funktion **strophe()** wird immer mit einem Wert in Klammern aufgerufen. So wird im Hauptprogramm mit **strophe(1)** die erste Strophe aufgerufen. Der Wert in den Klammern wird Argument genannt.
- Die Funktion **strophe()** muss ein Argument entgegennehmen, weshalb die Funktionsdefinition den Namen der Variable **stropheNum** enthält. Damit akzeptiert die Funktion diese Variable als Parameter.

```
<script>
function refrain() {
    var text = "... and they all go marching down \n";
    text += "to the ground \n";
    text += "to get out \n";
    text += "of the rain. \n";
    text += " \n";
    text += "boom boom boom boom boom boom boom boom \n";
    return text;
} // Ende Refrain

function strophe(stropheNum) {
    var kleineAmeise = "";
    if (stropheNum == 1) {
        kleineAmeise = "suck his thumb.";
    } else if (stropheNum == 2) {
        kleineAmeise = "rie his shoe.";
    } else {
        kleineAmeise = "I have no idea.";
    }

    var text = "The ants gor marching ";
    text += stropheNum + " by " + stropheNum + " hurra, hurrah \n";
    text += "The ants go marching ";
    text += stropheNum + " by " + stropheNum + " hurra, hurrah \n";
    text += "The ants go marching ";
    text += stropheNum + " by " + stropheNum;
    text += " the little one stops to ";
    text += kleineAmeise;
    return text;
} // Ende Strophe 1

// Hauptprogramm
alert(strophe(1));
alert(refrain());
alert(strophe(2));
alert(refrain());
</script>
```

FUNKTIONEN UND PARAMETER

Eine Funktion kann beliebig viele Parameter enthalten. Über die einzelnen Parameter können unterschiedliche Daten an die Funktion übertragen werden.

Erklärung der Funktion `refrain()`

- **Der Zweck der Funktion hat sich geändert.** Die Funktion hat nicht mehr die Aufgabe, einfach nur Text auf dem Bildschirm auszugeben. Stattdessen stellt sie nun dem Hauptprogramm Text zur Verfügung, und das Hauptprogramm kann damit machen, was es will.
- **Es gibt eine Variable mit dem Namen `text`.** Diese Variable übernimmt den gesamten Text, der an das Hauptprogramm gesendet wird.
- **Die Variable `text` wird über mehrere Zeilen hinweg verknüpft.** Hier wurde eine String-Verknüpfung gewählt, um einen aufwändigen Wert anzulegen. Achte auf die Verwendung des newline-Symbols (`\n`), mit dem eine neue Zeile in der Ausgabe begonnen wird. Das Zeichen erzwingt also einen Zeilenumbruch.
- **Die Anweisung `return` gibt `text` an das Hauptprogramm zurück.** Beachte, dass `return` die letzte Zeile der Funktion bilden soll.

```
<script>
  function refrain() {
    var text = "... and they all go marching down \n";
    text += "to the ground \n";
    text += "to get out \n";
    text += "of the rain. \n";
    text += " \n";
    text += "boom boom boom boom boom boom boom boom \n";
    return text;
  } // Ende Refrain
```

Die Begriffe Parameter und Argument werden häufig synonym verwendet.

ERKLÄRUNG DER FUNKTION STROPHE ()

- **Flexiblere Funktion.**
Diese Funktion kann verwendet werden, um viele verschiedene Strophen herzustellen
- **Sie nimmt sich eine Eingabe, um herauszufinden, welche Strophe ausgegeben werden muss.**
Der Ablaufplan sendet der Funktion eine Strophennummer.
- **Anhand der Eingabe modifiziert sie die Strophe.**
Die Strophennummer legt fest, wie der Rest der Strophe aussieht.
- **Sie gibt wie `refrain()` einen Wert zurück.**
Die Ausgabe dieser Funktion wird an das Hauptprogramm zurückgegeben, damit dieses etwas mit der neu geformten Strophe macht.

Die `if-else` in der `strophe`-Funktion

- Anfangs bleibt die Variable `kleineAmeise` leer. Wenn `stropheNum` den Wert **1** angenommen hat, erhält `kleineAmeise` den Wert `suck his thumb`, für Wert **2**, erhält die Variable den Wert `tie his shoe`. Jeder andere Wert wird als Fehler behandelt.

```
function strophe(stropheNum) {  
  var kleineAmeise = "";  
  if (stropheNum == 1) {  
    kleineAmeise = "suck his thumbn.";  
  } else if (stropheNum == 2) {  
    kleineAmeise = "rie his shoe.";  
  } else {  
    kleineAmeise = "I have no idea.";  
  }  
}
```


GÜLTIGKEITSBEREICHE



Funktionen ähneln in vielerlei Hinsicht Miniprogrammen. Jede innerhalb einer Funktion definierte Variable wirkt sich nur in der Funktion aus. Wenn die Ausführung der Funktion beendet ist, ist sie wieder verschwunden und undefiniert!

LOKALE UND GLOBALE VARIABLEN



Variablen auf der Hauptebene des Skripts werden globale Variablen genannt. Globale Variablen werden auf der hauptebene und innerhalb aller Funktionen bereitgestellt. Lokale Variablen existieren hingegen nur innerhalb der Funktion, für die sie definiert wurden. Wenn von Unterschieden zwischen lokalen und globalen Variablen die Rede ist, geht es um das Konzept der unterschiedlichen Gültigkeitsbereiche.

Variablen sollten so lokal wie möglich definiert werden. Globale Variablen werden nur dann benötigt, wenn Daten in mehreren Funktionen genutzt werden sollen.

EIN BASISARRAY

- Ein Array ist eine benannte Gruppe von Variablen. Es ist eine besondere Art von Variablen, die auch Feld genannt wird. Verwende ein Array, wenn mit einer Liste von Werten desselben Datentyps gearbeitet wird.
- Bei der Variable `art` handelt es sich um eine besondere Variable, weil sie mehrere unterschiedliche Werte enthält. Das Konstrukt `new Array(5)` reserviert im Arbeitsspeicher Platz für fünf Variablen, die alle `art` heißen.
- Jedes Element im Array wird über eine Ganzzahl identifiziert, die als Index dient. Die Zählweise beginnt normalerweise mit `0`.
- `art[0] = "Fußball";`
Diese Zeile bedeutet: Weise der Arrayvariablen `art` an Position `0` den Textwert `Fußball` zu.

```
<script>
  // Ein leeres Array anlegen
  var art = new Array(5);

  // Daten im Array ablegen
  art[0] = "Fußball";
  art[1] = "Baller";
  art[2] = "Karten";
  art[3] = "Action";
  art[4] = "Strategie";

  // Daten aus dem Array zurückgeben
  alert("Ich liebe " + art[4] + "spiele.");
</script>
```

ARRAYS MIT "FOR"-SCHLEIFEN VERWENDEN

```
<script>
  // ein Array vorladen
  var spieleListe = new Array("Flight Gear", "Sauerbraten", "Future Pinball", "Rac
er", "TORCS", "Orbiter", "Step Mania", "NetHack", "Marathon", "Crimson Fields");
  var text = "";
  for (i = 0; i < spieleListe.length; i++) {
    text += "Ich mag " + spieleListe[i] + ". \n";
  } // ende for-Schleife
  alert(text);
</script>
```

Was enthält der Code

- **Er enthält ein Array mit dem Namen `spieleListe`.** Dieses Array enthält die Namen einiger Spiele.
- **In das Array werden Daten vorgeladen.** Wenn beim Anlegen des Arrays gleich eine Liste mit Daten mitgegeben wird, lädt JavaScript diese Werte sofort in das Array vor. Bei vorgeladenen Werten muss die Größe des Arrays nicht angegeben werden.
- **Eine `for`-Schleife durchläuft das Array.** Arrays und `for`-Schleifen sind von Haus aus Partner. Die `for`-Schleife durchläuft jedes Element des Arrays.
- **Die Länge des Arrays (`spieleListe.length`) wird in der Bedingung der `for`-Schleife verwendet.** Statt die Länge fest mit einem Wert anzugeben, wurde in der `for`-Schleife die Eigenschaft `length` des Arrays verwendet. Die Schleife wird dadurch automatisch an die Größe des Arrays angepasst, sobald Elemente hinzugefügt oder entfernt werden.
- **Mit den einzelnen Elementen geschieht etwas.** Da `i` von 0 bis 9 läuft (und dies gleichzeitig die Indizes des Arrays sind), kann so problemlos jeder Wert des Arrays ausgegeben werden.

DAS LIEDBEISPIEL MIT ARRAY

- 💡 Es gibt ein Array mit dem Namen **listeAblenkung**. Der erste Eintrag ist leer, damit die Strophenummer mit der Liste übereinstimmt.
- 💡 Die Funktion **strophe()** kümmert sich um die **Ablenkung**. Da sich die Ablenkungen nun in einem Array befinden, kann **stropheNum** als Index verwendet werden, um eine bestimmte Ablenkung der kleinen Ameise in die Schleife zu übernehmen.
- 💡 Das Hauptprogramm ist eine **Schleife**. Jedes Element des Arrays **listeAblenkung** wird durchlaufen und die passende Strophe mit ihrem Refrain ausgegeben.
- 💡 Die Funktion **refrain()** wurde nicht geändert.

```
<script>
  var listeAblenkung = new Array("", "suck his thumb", "tie his shoe");

  function refrain() {
    var text = "... and they all go marching down \n";
    text += "to the ground \n";
    text += "to get out \n";
    text += "of the rain. \n";
    text += " \n";
    text += "boom boom boom boom boom boom boom boom \n";
    return text;
  } // Ende Refrain

  function strophe(stropheNum) {
    var kleineAmeise = listeAblenkung[stropheNum];

    var text = "The ants gor marching ";
    text += stropheNum + " by " + stropheNum + " hurra, hurrah \n";
    text += "The ants go marching ";
    text += stropheNum + " by " + stropheNum + " hurra, hurrah \n";
    text += "The ants go marching ";
    text += stropheNum + " by " + stropheNum;
    text += " the little one stops to ";
    text += kleineAmeise;
    return text;
  } // Ende Strophe 1

  // Hauptprogramm ist nun eine Schleife
  for (stropheNum = 1; stropheNum < listeAblenkung.length; stropheNum++) {
    alert(strophe(stropheNum));
    alert(refrain());
  } // Ende for-Schleife
</script>
```

EIGENE OBJEKTE ERSTELLEN

- Objekte besitzen normalerweise Eigenschaften und Methoden. Eine Eigenschaft ist wie eine Variable, die einem Objekt zugewiesen wird. Sie beschreibt das Objekt. Eine Methode ist wie eine Funktion, die aber zu einem Objekt gehört. Sie beschreibt Dinge, die das Objekt machen kann.



Mit Funktionen können Codeabschnitte zusammengefasst werden, Arrays erlauben es Variablen zusammenzufassen, bei Objekten kann sowohl Code als auch Variablen (also Funktionen und Variablen und Arrays) in einem großen Konstrukt zusammengefasst werden.

OBJEKT ERSTELLEN

💡 Ein neues Objekt wird angelegt.

Über die Syntax `new Object()` wird ein neues Objekt angelegt.

💡 Eigenschaften hinzufügen.

Eine Eigenschaft ist wie eine Subvariable. Sie ist nicht mehr als eine Variable, die an ein spezielles Objekt gebunden ist. Wenn man beispielsweise `kind.name` einen Wert zuweist, wird damit festgelegt, dass `kind` eine Eigenschaft besitzt, die `name` heißt. Außerdem wird der Eigenschaft ein Anfangswert zugewiesen.

💡 Ein Objekt kann beliebig viele Eigenschaften besitzen.

Damit können Variablen zu größeren Objekten zusammengefasst werden.

💡 Jede Eigenschaft kann jeden beliebigen Datentyp enthalten.

Anders als bei Arrays, bei denen es üblich ist, dass alle Elemente von genau demselben Datentyp sind, kann jede Eigenschaft einen anderen Datentyp enthalten.

💡 Verwendung der Syntax mit dem Punkt, um Eigenschaften zu betrachten oder zu ändern.

Wenn das Objekt `kind` eine Eigenschaft `name` besitzt, kann `kind.name` wie eine Variable verwendet werden.

```
<script>

//Objekt "Kleinkind" erstelle
var kind = new Object();

//Eigenschaften hinzufügen
kind.name = "Milo";
kind.alter = 5;

// Die Werte der eigenschaften zeigen
alert("Der Name des Kindes lautet " + kind.name + ".");

</script>
```

OBJEKTE UND EIGENSCHAFTEN UND METHODEN

- › Objekte besitzen zusätzlich zu Eigenschaften noch weitere Merkmale. Zudem können sie auch Methoden besitzen. Bei einer Methode handelt es sich einfach nur um eine mit einem Objekt verbundene Funktion.

```
<script>
  //Das Kind anlegen
  var kind = new Object();

  //Eigenschaften hinzufügen
  kind.name = "Milo";
  kind.alter = 5;

  //Methode erstellen
  kind.sagt = function() {
    info = "Hallo! Mein Name ist " + this.name;
    info += " und ich bin " + this.alter + ".";
    alert(info);
  } //Ende Methode

  // Die Methode sagt aufrufen
  kind.sagt();
</script>
```


WIEDERVERWENDBARE OBJEKTE – KONSTRUKTOREN

- Zu diesem Beispiel gehört das Erstellen einer Klasse (einem Muster für das Erstellen von Objekten) und das Wiederverwenden dieser Definition, um zwei Kinder anzulegen.
- **Festlegen einer ganz normalen Funktion.**
In JavaScript werden Klassen als Erweiterungen von Funktionen definiert. Der Name einer Funktion ist gleichzeitig der Name der Klasse. Beachte, dass der Name einer Klassenfunktion normalerweise mit einem Großbuchstaben beginnt. Wenn eine Funktion verwendet wird, um ein Objekt zu beschreiben, wird diese Funktion auch als Konstruktor des Objektes bezeichnet. Wenn man es wünscht, kann der Konstruktor Parameter übernehmen, aber er gibt normalerweise keine Werte zurück. In diesem Beispiel sind Parameter für den Namen und das Alter dabei.
- **Verwendung von `this`, um Eigenschaften anzulegen**
Füge beliebige Eigenschaften einschließlich Standardwerten hinzu. Beachte, dass diese Werte später beliebig geändert werden können. Jede Eigenschaft sollte mit `this` und einem Punkt anfangen. In diesem Beispiel werden lokale Parameter verwendet, um die Eigenschaften festzulegen.
- **Verwendung von `this`, um Methoden zu definieren.**
Wenn das Objekt eine Methode besitzen soll, wird das definiert, indem der Operator `this` verwendet wird, dem das Schlüsselwort `function()` folgt. Es können beliebig viele Funktionen hinzugefügt werden.

```
<script>
  // Konstruktor anlegen
  function Kind(lname, lalter) {
    this.name = lname;
    this.alter = lalter;
    this.info = function() {
      nachricht = "Hallo! Mein Name ist " + this.name;
      nachricht += " und ich bin " + this.alter + ".";
      alert(nachricht);
    } //Ende Methode info
  } // Ende Klassendefinition Kind

  function zentral() {
    // zwei Kinder anlegen
    kindA = new Kind("Alpha", 1);
    kindB = new Kind("Beta", 2);
    kindB.name = "Charlie";
    kindB.alter = 3;

    //die Kinder informieren
    kindA.info();
    kindB.info();
  } // Ende zentral
  zentral();
</script>
```

DIE FUNCTION ZENTRAL ()

💡 **Achte darauf, ob man auf die Klasse zugreifen kann.**

Eine Klasse kann erst dann sinnvoll eingesetzt werden, wenn JavaScript sie kennt. In diesem Beispiel wird die Klasse im Code definiert.

💡 **Mit dem Schlüsselwort `new` eine neue Instanz der Klasse anlegen.**

Das Schlüsselwort `new` besagt, dass auf der Grundlage der Definition ein besonderes `kind` "gemacht" werden soll. Normalerweise wird dies einer Variable zugewiesen. Der Konstruktor erwartet den Namen und das Alter; daher legt er automatisch ein `kind` mit den Vorgaben Name und Alter an.

💡 **Änderung der Eigenschaften der Klasse**

Man kann die Werte aller Eigenschaften der Klasse ändern. In dem Beispiel wird bei dem Namen und dem Alter des zweiten Kindes die Eigenschaft geändert.

💡 **Aufrufen der Methode der Klasse.**

Da die Klasse `Kind` die Methode `info()` besitzt, kann sie jedes Mal darauf zugreifen, wenn ein bestimmtes `kind` die Benutzer informieren soll.

```
function zentral() {  
  // zwei Kinder anlegen  
  kindA = new Kind("Alpha", 1);  
  kindB = new Kind("Beta", 2);  
  kindB.name = "Charlie";  
  kindB.alter = 3;  
  
  //die Kinder informieren  
  kindA.info();  
  kindB.info();  
} // Ende zentral
```

ALLGEMEINER HINWEIS



- Die Art, wie JavaScript Objekte anlegt und verwendet, ist einfach, entspricht aber nicht unbedingt dem Standard. Die meisten Sprachen, die objektorientierte Techniken unterstützen, machen dies anders. Die Vererbung wird von JavaScript auch nicht wie sonst üblich unterstützt. JavaScript kennt dafür Prototyping (oder Mustererstellung).

EINFÜHRUNG IN JSON

- Objekte und Arrays von JS sind unglaublich flexibel einsetzbar. Aufgrund ihrer Leistungsstärke und einfachen Bedienung sind sie so bekannt geworden, dass ein spezielles Datenformat, JSON (JavaScript Objekt Notation), von vielen anderen Sprachen übernommen wurde.
- JSON wird hauptsächlich verwendet, um komplexe Daten (insb. multidimensionale Arrays) zu speichern und diese von einem Programm an ein anderes Programm weiterzugeben. Für JSON-Daten werden keine Konstruktoren benötigt, weil die Daten selbst verwendet werden, um die Struktur der Klasse festzulegen.
- Vor allem über AJAX und jQuery-Bibliotheken wird JSON gerne intensiv genutzt, um komplexe Daten auf einfache Art zu verwalten.

DATEN IM JSON -FORMAT ABLEGEN

- Dieser Code beschreibt ein **kind** mit zwei Eigenschaften: einem Namen und einem Alter. **kind** sieht fast wie ein Array aus, aber statt numerischer Inhaltsangaben verwendet **kind** für die Erkennung String-Werte. Darüber hinaus handelt es sich dabei um ein Objekt.
- es gibt zwei Verschiedene Möglichkeiten auf die Einträge zuzugreifen

```
alert(kind["name"]); // Array-Syntax  
alert(kind.alter); // Objekt-Syntax
```

```
<script>  
  var kind = {  
    "name": "George",  
    "alter": 10  
  };  
</script>
```

- Die meisten internen JavaScript-Objekte verwenden die Punkt-Notation, aber beide Lösungen sind gültig

```
<script type = "text/javascript">  
  var kind = {  
    "Name": "George",  
    "Alter": 10  
  };  
  
  alert(kind.Name);  
  
  for (schluessel in kind){  
    alert(schluessel+ ": " + eval ("kind." + schluessel));  
  }  
</script>
```

JSON ABSPEICHERN

- Auf folgende Weise werden Daten im JSON-Format gespeichert:
 - ▶ **Lege die Variable an.**
Du kannst wie bei jeder Variable die Anweisung **var** verwenden
 - ▶ **Erfasse den Inhalt in geschweiften Klammern ({})**
Hierbei handelt es sich um dieselbe Technik wie beim Erstellen vorgeladener Arrays
 - ▶ **Lege einen Schlüssel an**
Bei **kind** sollten die Eigenschaften **"Name"** und **"Alter"** heißen - was bedeutet, dass die Indexkennung über Wörter und nicht über Zahlen erfolgt. Bei jeder Eigenschaft beginne ich mit deren Namen. Der Schlüssel kann eine Zeichenkette (ein String) oder eine Ganzzahl (ein Integerwert) sein.
 - ▶ **Dem Schlüssel folgt ein Doppelpunkt (:)**
Dem Schlüssel folgt unmittelbar ein Doppelpunkt
 - ▶ **Legen den Wert an, der dem Schlüssel zugeiwesen wird**
Du kannst einen Schlüssel jeden beliebigen Wertetyp zuweisen.
In diesem Fall wurde dem Schlüssel **"Name"** den Wert **"George"** zugewiesen
 - ▶ **Trennen die einzelnen Namen/Werte-Paare durch ein Komma (,)**
Du kannst so beliebig viele Namen/Wert-Paare hinzufügen

KOMPLEXERE JSON-STRUKTUREN

- › zweidimensionales Array
Unterschied zum vorangegangenen BSP:
 - ▶ **entfernung ist ein JSON-Objekt.** Die gesamte Datenstruktur wird in einer einzigen Variablen abgelegt. Diese Variable ist ein JSON-Objekt mit Namen/Wert-Paaren
 - ▶ **Das Objekt entfernung hat vier Schlüssel.** Diese stimmen mit den vier Zeilen der ursprünglichen Tabelle überein
 - ▶ **Bei den Schlüsseln handelt es sich um Städtenamen.** Das ursprüngliche 2D-Array verwendet numerische Indizes. Diese sind zwar bequem, aber auch ein wenig künstlich. In der JSON-Struktur bilden die Städtenamen die Indizes.
 - ▶ **Der Wert eines Eintrags ist ein JSON-Objekt.** Der Wert eines JSON-Elements kann alles sein - auch ein anderes JSON-Objekt. Damit können wir wirklich komplexe Beziehungen in einer einzigen Variablen zusammengefasst werden
 - ▶ **Jede Zeile kann als JSON Objekt zusammengefasst werden.** So ist beispielsweise der Wert, der zu Indianapolis gehört, eine Liste mit Entfernungen zwischen **Indianapolis** und den verschiedenen Städten
 - ▶ **Die gesamte Definition bildet im Code nur eine "Zeile".** Obwohl sie im Editor (aus Gründen der Übersichtlichkeit) mehrere Zeilen einnimmt, besteht die Gesamte Definition in Wirklichkeit nur aus einer einzigen Codezeile.

```
<script>
var entfernung = {
  "Indianapolis" :
    { "Indianapolis": 0,
      "New York": 648,
      "Tokyo": 6476,
      "London": 4000 },


  "New York" :
    { "Indianapolis": 648,
      "New York": 0,
      "Tokyo": 6760,
      "London": 3470 },

  "Tokyo" :
    { "Indianapolis": 6476,
      "New York": 6760,
      "Tokyo": 0,
      "London": 5956 },

  "London" :
    { "Indianapolis": 4000,
      "New York": 3470,
      "Tokyo": 5956,
      "London": 0 },
};
alert(entfernung["Indianapolis"]["London"]);
alert(entfernung.Indianapolis.Tokyo);
alert(entfernung["London"].Tokyo);
</script>
```

JSON BESITZT ALS DATENFORMAT EINE REIHE WICHTIGER VORTEILE

- **Es dokumentiert sich selbst.** Selbst wenn du nur die Datenstruktur ohne weiteren Code siehst, kannst du ihre Bedeutung verstehen
- **Die Verwendung von Strings für die Inhaltsangabe macht den Code lesbarer.**
`entfernung["Indianapolis"]["London"]` ist leichter zu verstehen als `entfernung[0][3]`
- **JSON Daten können als Text gespeichert und übertragen werden.** Es hat sich gezeigt, dass dies tiefgreifende Auswirkungen auf die Webprogrammierung hat. Dies trifft besonders auf AJAX zu.
- **JSON kann komplexe Beziehungen beschreiben.** Bei dem hier vorgestellten Beispiel handelt es sich um ein einfaches zweidimensionales Array, aber das JSON-Format kann auch zur Beschreibung komplexerer Beziehungen und zur Datenbankunterstützung eingesetzt werden.
- **Viele Sprachen unterstützen das JSON-Format.** Viele Websprachen besitzen inzwischen eine direkte JSON-Unterstützung. Die wichtigste davon ist die Sprache PHP, die sehr häufig in AJAX-Anwendungen zusammen mit JavaScript verwendet wird.
- **JSON ist kompakter als XML.** Häufig wird das Datenformat XML für die Übermittlung komplexer Daten verwendet. Es gilt, dass JSON kompakter und nicht so "geschwätzig" ist wie XML
- **JavaScript kann von Haus aus mit JSON umgehen.** Einige Sprachen müssen JSON erst übersetzen, bevor sie es nutzen können. Anders JS.
- **In JSON Objekte können auch Methoden eingebettet werden.** Ist aber eigentlich unüblich, wenn es um den Datentransport mit JSON geht.



Quelle: JavaScript für Dummies

ISBN: 978-3-527-71444-5
(Keine Kaufempfehlung!)

