

**Begleit-Skriptum**

# **JavaScript 1.8**

**(Herdt Buch)**

**2. Klasse | Medientechnik**



**Christian Haabs**

# Inhaltsverzeichnis

Kapitel 2 „Einführung in JavaScript“ .....	4
Was ist JavaScript? .....	4
Was kann man mit JavaScript tun? .....	4
Das erste JavaScript.....	4
Die JavaScript Console.....	5
Wiederholungsfragen zu Kapitel 2 „Einführung in JavaScript“ .....	5
Übungsbeispiele zu Kapitel 2 „Einführung in JavaScript“ .....	6
Kapitel 3 „Grundlegende Sprachelemente“ .....	7
Wiederholungsfragen zu Kapitel 3 „Grundlegende Sprachelemente“.....	7
Übungsbeispiele zu Kapitel 3 „Grundlegende Sprachelemente“ .....	8
Variablen – Konstanten .....	8
Datentypen – Operatoren .....	8
Kapitel 4 „Kontrollstrukturen“ .....	9
Wiederholungsfragen zu Kapitel 4 „Kontrollstrukturen“ .....	9
Übungsbeispiele zu Kapitel 4 „Kontrollstrukturen“ .....	10
Verzweigungen .....	10
Schleifen .....	10
Kapitel 5 „Funktionen“ .....	12
Werte aus einem Formular an eine Funktion übergeben .....	12
Wiederholungsfragen zu Kapitel 5 „Funktionen“.....	13
Übungsbeispiele zu Kapitel 5 „Funktionen“ .....	14
Kapitel 6 „Objekte“ .....	15
Benutzerdefinierte Objekte in JavaScript.....	15
Wiederholungsfragen zu Kapitel 6 „Objekte“ .....	16
Übungsbeispiele zu Kapitel 6 „Objekte“.....	17
Eigenschaften .....	17
Methoden .....	17
Kapitel 7 „Vordefinierte Objekte“ .....	18
Das oberste Objekt „Object“ .....	18
Objekte für die primitiven Datentypen Number, String und Boolean .....	18
„Math“-Objekt .....	19
Reguläre Ausdrücke („RegExp“-Objekte) .....	19
„Array“-Objekt.....	19
Wiederholungsfragen zu Kapitel 7 „Vordefinierte Objekte“ .....	20
Übungsbeispiele zu Kapitel 7 „Vordefinierte Objekte“ .....	20
Objekte für die primitiven Datentypen Number, String und Boolean .....	20

„Math“-Objekt und „RegExp“-Objekte.....	21
„Array“-Objekt.....	21
Kapitel 8 „DOM-Konzept – Browser-Objekte“ .....	22
Wiederholungsfragen zu Kapitel 8 „DOM-Konzept – Browser-Objekte“ .....	22
Übungsbeispiele zu Kapitel 8 „DOM-Konzept – Browser-Objekte“ .....	23
Window-Objekt .....	23
Document-Objekt .....	23
History-Objekt .....	23
Location-Objekt .....	23
Screen-Objekt .....	23
Navigator-Objekt .....	23
Kapitel 9 „Ereignisse“ .....	24
Wiederholungsfragen zu Kapitel 9 „Ereignisse“ .....	24
Übungsbeispiele zu Kapitel 9 „Ereignisse“ .....	25
Abschließendes Beispiel .....	26

## Kapitel 2 „Einführung in JavaScript“

### Was ist JavaScript?

- JavaScript wurde entwickelt, um Interaktivität in HTML Webseiten zu ermöglichen.
- JavaScript ist eine Skriptsprache. Eine Skriptsprache ist eine leichtgewichtige Programmiersprache.
- JavaScript wird direkt in HTML Webseiten eingebettet.
- JavaScript ist eine interpretierte Sprache (d.h. sie muss nicht zuvor kompiliert werden).
- JavaScript kann von jedem ohne Kauf einer Lizenz verwendet werden.
- JavaScript ist case-sensitiv.

### Was kann man mit JavaScript tun?

- JavaScript stellt Designern ein ProgrammierTool zur Verfügung - HTML-Designer sind zumeist keine Programmierer, JavaScript ist jedoch eine Skriptsprache mit sehr einfacher Syntax. JavaScript ist somit für jeden einfach zu erlernen.
- JavaScript kann Dynamik in eine HTML Webseite bringen. Mit JavaScript können Sie variablen Text in eine HTML Webseite schreiben, z.B. "<h1>" + name + "</h1>".
- JavaScript kann auf Events reagieren. Sie können beispielsweise mit JavaScript eine Aktion auslösen, sobald ein Benutzer auf einen Button klickt.
- JavaScript kann HTML Elemente lesen und schreiben. Mit JavaScript können Sie die Eigenschaften eines HTML Elements verändern.
- JavaScript kann zur Validierung von Daten verwendet werden. Mit JavaScript können Sie Formulardaten vor dem Senden zum Web Server überprüfen.
- JavaScript kann verwendet werden, um den aktuellen Web Browser des Benutzers zu bestimmen. Mit JavaScript können Sie Browserweichen erstellen.
- JavaScript ermöglicht das Erzeugen von Cookies. Ein JavaScript kann Informationen am Computer des Benutzers speichern und lesen.

### Das erste JavaScript

Die Ausgabe von „Hello World“ zählt zu den ersten Schritten beim Erlernen einer neuen Programmier- oder Skriptsprache.

```
<html>
  <head><title>Hello World</title></head>
  <body>
    <script type="text/javascript">
      document.write("Hello World!");
    </script>
    <noscript>
      Your Web Browser does not support JavaScript!
    </noscript>
  </body>
</html>
```

Das Semikolon am Ende einer Zeile schließt einen einzelnen Befehl in JavaScript ab. JavaScript ist **case-sensitiv**, das bedeutet, dass JavaScript zwischen Groß- und Kleinschreibung unterscheidet.

## Die JavaScript Console

Beim Entwickeln mit JavaScript können Programmierfehler leicht übersehen werden. Als Unterstützung zur Fehlersuche kann die JavaScript Console (zu finden in z.B. Firefox: Extras – Web Entwickler – Browser-Konsole) verwendet werden.

```
...
<script type="text/javascript">
    document.write("Hello World!");
    /* output a simple message to the console */
    console.log("hello world works fine.");
    /* cause an error and have a look at the console */
    window.alert(no quotation marks are set);
</script>
...
```

## Wiederholungsfragen zu Kapitel 2 „Einführung in JavaScript“

1. Welche Firma hat JS entwickelt? Wer ist für die Weiterentwicklung verantwortlich?
2. Wie lautet die korrekte Bezeichnung von JS?
3. Was ist eine RIA? Nenne ein Beispiel!
4. Zähle einige der möglichen Einsatzgebiete von JS auf!
5. Was ist eine Browserweiche?
6. Ist JS eine objektorientierte oder eine prozedurale Skriptsprache?
7. Was bedeutet JS ist „case-sensitiv“?
8. Wird JS client- oder serverseitig ausgeführt?
9. Wie wird JS in einer Webseite eingebunden?
10. Was passiert, wenn ein Browser JS deaktiviert hat?
11. Notiere den korrekten JS-MIME-Type!
12. Wie gefährlich ist JS?
13. Was bedeutet „JS-Anwendungen laufen in einer Sandbox ab“?
14. Was muss im IE aktiviert sein, damit JS-Anwendungen laufen?
15. Welche Skriptsprache wird von Microsoft im IE verwendet?
16. Wie lautet die aktuelle Version von JS?
17. Wie kann man Browser, die eine bestimmte JS-Version nicht unterstützen, vom verwendeten Script-Bereich fernhalten?
18. Warum sollte das language-Attribut zur Versionsdefinition benutzt werden?
19. Warum wird in der Praxis auf die Angabe einer JS-Version verzichtet?
20. Erkläre die Unterschiede der Sprachen VBScript, JScript, Java und JavaScript!
21. Wozu dient die JS-Console und wie kannst du sie nutzen?

## Übungsbeispiele zu Kapitel 2 „Einführung in JavaScript“

1. Schreibe ein erstes JavaScript das „Hello World!“ ausgibt!  
Verwende dazu Notepad++ als Editor.  
Verwenden zur Ablage deiner Dateien folgende Struktur:
  - a) Lege einen eigenen Ordner mit Nachnamen und Vornamen (Bsp.: Mueller\_Karl) für die Lösung der Aufgaben an.
  - b) Erstelle darin eine Seite „index.html“ in der deine Daten (Klasse, Name) angeben sind und wo auf jede der nachfolgenden Aufgaben verlinkt wird.
  - c) Jede Aufgabe ist in einer eigenen Datei abzuspeichern.
  - d) Vermerke auch die Angabe aus diesem Skript zu jeder Aufgabe!
2. Verändere das „Hello World“-Skript derart, dass es folgendes ausgibt: „Hello <Dein Name>!“. Erweiterung: Formatiere die Ausgabe in JavaScript mit dem <h1>-Tag!
3. Schreibe ein JS, das überprüft, welche JS-Version der benutzte Browser unterstützt.

*Tipp: Nutze das language-Attribut!*

```
<script language="javascript1.1">
    document.write("Teste auf JavaScript Version 1.1... Ok!<br>");
</script>
```

4. Teste die JS-Console mit obigem Beispiel!  
Bessere eventuelle Fehler im Skript aus und teste erneut!

## Kapitel 3 „Grundlegende Sprachelemente“

JavaScript ist objektbasiert. Ein Programm ist eine Folge von Befehlen, deren Abarbeitung durch Kontrollstrukturen steuerbar ist. In Ausdrücken werden Operanden mit Operatoren verknüpft. JavaScript unterscheidet Groß- und Kleinschreibung, ist also case-sensitiv. JavaScript gilt als schwach typisierte Sprache.

Der JS-Code kann sowohl im head-Bereich, als auch im body-Bereich mit dem `<script>`-Tag eingefügt werden. JS-Code kann man auch als separate Datei (Endung .js) einbinden.

### Wiederholungsfragen zu Kapitel 3 „Grundlegende Sprachelemente“

1. Welche grundsätzlichen Möglichkeiten gibt es JS in HTML-Seiten einzubinden?
2. In welchen HTML-Bereichen kann JS direkt eingebunden werden?
3. Muss in HTML5 der MIME-Typ verpflichtend angegeben werden?
4. Kann es in einer HTML-Seite mehrere JS-Skriptbereiche geben?
5. Wo und wie werden JS-Code als externe Datei eingebunden?
6. Welche Vorteile können dadurch erzielt werden?
7. Welche Extention muss eine JS-Datei haben?
8. Wozu dienen HTML-Eventhandler in JS? Gib ein Beispiel!
9. Was versteht man in JS unter einer Inlinerefenz? Gib ein Beispiel!
10. Warum sollten HTML-Eventhandler und Inlinerefrenzen nicht mehr verwendet werden?
11. Was ist eine geeignete Reaktion falls JS in einem Browser nicht verfügbar ist?
12. Wodurch wird eine Anweisung in JS beendet?
13. Wie werden Kommentare in JS notiert?
14. Was sind Schlüsselwörter?
15. Was sind Bezeichner unter JS? Welche Zeichen sind erlaubt?
16. Wie werden in JS Variablen definiert?
17. Was versteht man unter „Loser Typisierung“?
18. Wie werden unter JS Konstanten definiert? Was ist daran problematisch?
19. Wie wird unter JS einer Variablen ein Datentyp zugewiesen?
20. Was versteht man unter „Casting“? Ist es unter JS notwendig?
21. Welche Datentypen sind unter JS verfügbar?
22. Wie werden Strings unter JS gekennzeichnet?
23. Wie unterscheidet sich die Funktion des +-Operators bei Zahlen bzw. Strings?
24. Wie wird unter JS der Zeilenumbruch in Zeichenketten kodiert?
25. Was sind arithmetische Operatoren? Nenne einige Beispiele?
26. Was sind Vergleichsoperatoren? Nenne einige Beispiele?
27. Was sind logische Operatoren? Nenne einige Beispiele?
28. Wie kann man unter JS den aktuellen Typ einer Variablen auslesen?

# Übungsbeispiele zu Kapitel 3 „Grundlegende Sprachelemente“

W3Schools Tutorial: JS HOME bis JS Comments und JS Variables bis JS Comparisons

## Variablen – Konstanten

1. Teste das Einbinden einer externen JS-Datei in eine HTML-Datei.  
Es soll der Text „Ich bin ein Text aus einer externen JS-Datei!“ angezeigt werden.  
Definiere in der Datei auch eine Variable *Pi*, weise den entsprechenden Wert zu und veranlasse eine Ausgabe.
2. Es ist ein JS zu erstellen, das einen beliebigen Text ausgibt und mindestens drei HTML-Tags zur Formatierung verwendet, zB. Überschrift1, Zeilenumbruch und Absatz.
3. Es ist ein JS zu erstellen, das die Variable `name` (enthält deinen Namen als Text) als Überschrift1 formatiert ausgibt.
4. Es ist ein JS zu erstellen, das die Variablen `a="100"+44` und `b="100"-44` definiert und ausgibt.  
Welcher Unterschied ist festzustellen? Worauf ist das zurückzuführen?

## Datentypen – Operatoren

5. Es ist ein JS zu erstellen, das die Anzahl der Beine von fünf Schafen berechnet und ausgibt. Die Variable `name:=Schafe` und die Variable `anzahl:=5` sind dabei zu verwenden.
6. Es ist ein JS zu erstellen, das das Alter von Donald Duck berechnet (`name:=Donald Duck` und `geburtsjahr:=1954`) und folgenden Satz formatiert ausgibt: „Donald Duck ist heuer ? Jahre alt“.  
**Hinweis:** `datum=new Date(); var jahr=datum.getFullYear();`
7. Schreibe ein Skript, das folgende Wahrheitstabelle unter Verwendung von Zeichenketten und booleschen Variablen ausgibt:

Wahrheitstabelle

UND	true	false
true	true	false
false	false	false

ODER	true	false
true	true	true
false	true	false

8. Schreibe ein Skript, das Variable `a` die Zahl `3`, `b` den Text „Hallo Welt“ und `c` den Wahrheitswert „`true`“ zuweist. Lies nun die Typen der einzelnen Variablen aus und zeige sie an.

## Kapitel 4 „Kontrollstrukturen“

Anweisungen sind die kleinste ausführbare Einheit eines JS-Programms. Kontrollstrukturen zur Auswahl und Wiederholung sind an Bedingungen geknüpft und stellen Verzweigungen und Schleifen dar.

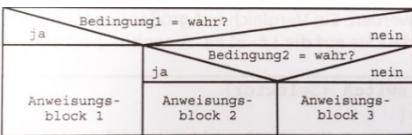
### Bedingte Anweisungen

- If-else-Verzweigung
- Switch-Anweisung

### Schleifen

- Zählschleife
- While-Schleife
- Do-while-Anweisung

## Wiederholungsfragen zu Kapitel 4 „Kontrollstrukturen“

1. Was ist die kleinste ausführbare Einheit eines JS-Programms?
2. Welche Möglichkeiten der bedingten Ausführung kennst du?
3. Mit welchem Schlüsselwort wird eine einfache Bedingung eingeleitet?
4. Kodiere:  

5. Mit welchen Schlüsselwörtern kann eine mehrseitige Bedingung umgesetzt werden?
6. Über welches Schlüsselwort kann die Ausführung einer mehrseitigen Bedingung beendet werden?
7. Wann ist die Verwendung einer zählergesteuerten Schleife sinnvoll?
8. Mit welchem Schlüsselwort wird eine zählergesteuerte Schleife notiert?
9. Was sind kopfgesteuerte Schleifen und welches Schlüsselwort wird dazu verwendet?
10. Wird eine kopfgesteuerte Schleife jedenfalls mindestens 1 x durchlaufen?
11. Was sind fußgesteuerte Schleifen und welche Schlüsselwörter werden dazu verwendet?
12. Wird eine fußgesteuerte Schleife jedenfalls mindestens 1 x durchlaufen?
13. Wodurch unterscheiden sich die Schlüsselwörter break und continue in der Schleifensteuerung?
14. Was versteht man unter einer Endlosschleife?

## Übungsbeispiele zu Kapitel 4 „Kontrollstrukturen“

### Verzweigungen

#### 1. Begrüßung

Erstelle ein JS, das je nach Uhrzeit folgendes ausgibt:

vor 12 Uhr mittags: „Guten Morgen“

nach 12 Uhr mittags: „Guten Tag“

Hinweis: `var stunde=datum.getHours();`

#### 2. Auswertung einer Berechnung

Erweitere das Beispiel 6 aus Kapitel 3 und gib je nach Alter zusätzlich folgende Sätze formatiert aus:

„Er ist über 60 Jahre alt und damit in Pension.“,

„Er ist unter 60 Jahre alt und noch voll im Einsatz.“.

#### 3. Auswertung einer Berechnung

Es ist ein JS zu erstellen, das den aktuellen Wochentag ermittelt. Je nach Wochentag soll ausgegeben werden, welche Kleidung zu tragen ist.

Beispiel: „Heute ist Sonntag und ich empfehle Ihnen den Bademantel.“.

### Schleifen

#### 4. Zählschleife

Es ist ein JS zu erstellen, das in einer Schleife folgenden Text ausgibt:

Die aktuelle Zahl ist 0.

Die aktuelle Zahl ist 1.

Die aktuelle Zahl ist 2.

Die aktuelle Zahl ist 3.

Die aktuelle Zahl ist 4.

#### 5. Zählschleife 2

Es ist ein JS zu erstellen, das in einer Schleife folgenden Text samt entsprechender Formatierung ausgibt:

Überschrift 1

Überschrift 2

Überschrift 3

Überschrift 4

Überschrift 5

Überschrift 6

#### 6. Schleifen

Erstelle ein JS, das mit Hilfe von Schleifen eine Tabelle mit 3 Spalten und 10 Zeilen erzeugt (Hinweis: `border = 1` einstellen). Fülle die Tabelle mit den Zahlen 1 bis 30!

## 7. Kleines 1x1

Erstelle mit Hilfe von JS eine Tabelle mit 9 Zeilen und 9 Spalten, in denen das kleine Ein-mal-Eins dargestellt wird (zB. in Zeile 3, Spalte 7 soll das Ergebnis von  $3*7$  stehen).

**Das kleine Ein-mal-Eins**

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

## 8. Aktueller Monatskalender

Schreibe ein JS, das den aktuellen Monatskalender in einer Tabelle ausgibt. Der heutige Tag soll erkannt und fett geschrieben werden.

**Kalender für 9.2011**

Mo	Di	Mi	Do	Fr	Sa	So
				1	2	3
5	6	7	8	9	10	<b>11</b>
12	13	14	15	16	17	18
19	20	21	22	23	<b>24</b>	25
26	27	28	29	30		

Hinweise: Erzeuge eine Instanz des Date-Objekts mit dem aktuellen Datum. `date.setDate(1);` setzt den Tag des Monats auf "1". Nun kann bestimmt werden, auf welchen Wochentag der Erste des Monats fällt. Daraus ergibt sich, wie viele leere Felder am Anfang des Kalenders eingefügt werden müssen.

Um auch Monate mit weniger als 31 Tagen richtig darzustellen, kann folgende Eigenschaft des Date-Objekts benutzt werden: wenn das Datum mit `setDate()` zB. auf den 31.09. gesetzt wird, korrigiert JavaScript das Datum auf den 01.10. Nun kann man prüfen, ob der Monat des erzeugten Datums noch dem aktuellen Monat entspricht. Ist dies nicht der Fall, dann soll die Ausgabe vorzeitig abgebrochen werden.

### Ergänzung:

Ändere den Kalender in einen Advent-Kalender: Die einzelnen Tage sollen jeweils als Link auf eine beliebige Seite (oder ein Bild) angezeigt werden – allerdings nur für die Tage bis zum aktuellen Datum.

**Link-Kalender für 9.2011**

Mo	Di	Mi	Do	Fr	Sa	So
				1	2	3
5	6	7	8	9	<b>10</b>	11
12	13	14	15	16	17	18
19	20	21	22	23	<b>24</b>	25
26	27	28	29	30		

## Kapitel 5 „Funktionen“

Funktionen sind in sich abgeschlossene Programme, die Teilaufgaben lösen und innerhalb eines Programms Mehrfachverwendung finden.

Das Schlüsselwort für eine Funktion ist `function`, danach folgt der Name der Funktion mit einer Parameterliste in runden Klammern. Die Anweisungen einer Funktion werden in geschweiften Klammern eingebettet. Die Parameterliste ist optional.

Die Rückkehr aus einer Funktion erfolgt nach Abarbeitung aller Programmzeilen oder bei Erreichen der Anweisung `return`.

Eine Funktionsdeklaration wird von der Laufzeitumgebung als Erstes interpretiert, daher ist es egal, wo die Funktionsdeklaration im Quelltext steht.

In JavaScript gibt es mehrere Möglichkeiten, Funktionen (genauer: Funktionsobjekte) zu erzeugen.

### Bsp.: Funktionsdeklaration

```
function summe( parameter1, parameter2 ) {  
    // Code für die Funktion  
    var sum = parameter1 + parameter2;  
    return sum;  
}  
  
var gesamt = summe(200, 100); // Funktionsaufruf
```

### Bsp.: Funktionsausdruck

```
var summe = function ( parameter1, parameter2 ) {  
    return parameter1 + parameter2;  
}  
var gesamt = summe(200, 100); // Funktionsaufruf
```

## Werte aus einem Formular an eine Funktion übergeben

```
<form>  
    Zahl 1: <input type="number" id="zah11"><br>  
    <input type="button" onClick="berechnen()" value="Berechnen">  
</form>  
  
<script type="text/javascript">  
...  
    function berechnen () {  
        var a = document.getElementById("zah11");  
        document.write("Objekt von Zahl 1: ", a, "<br>");  
        document.write("Wert von Zahl 1: ", a.value, "<br>");  
        document.write("Zahl 1 + Zahl 1: ", a.value+a.value, "<br>");  
        var b = parseInt(a.value);  
        document.write("Zahl 1 + Zahl 1: ", b+b);  
    }  
...  
</script>
```

Zahl 1:

```
Objekt von Zahl 1: [object HTMLInputElement]  
Wert von Zahl 1: 5  
Zahl 1 + Zahl 1: 55  
Zahl 1 + Zahl 1: 10
```

## Wiederholungsfragen zu Kapitel 5 „Funktionen“

1. Welchen Zweck haben Funktionen in einem JS-Programm?
2. Mit welchem Schlüsselwort wird eine Funktion eingeleitet?
3. Dürfen sich Funktionsaufruf und Funktionsdefinition in verschiedenen <script>-Abschnitten befinden? Worauf ist dabei zu achten?
4. Wie kann eine Funktion aufgerufen werden?
5. Wie kann ein Wert aus einer Funktion zurück an den Aufrufer gegeben werden?
6. Wie können Werte aus einem Formular in einer Funktion verarbeitet werden?
7. Wie können Parameter an eine Funktion übergeben werden?
8. Was versteht man unter einer variablen Parameterliste?
9. Über welches Schlüsselwort kann auf diese Parameter zugegriffen werden?
10. Mit welcher Methode kann man die Anzahl der übergebenen Parameter ermitteln?
11. Wodurch unterscheiden sich lokale Variablen von globalen Variablen?
12. Wozu dienen die vordefinierten Funktionen parseFloat() und parseInt()?

## Übungsbeispiele zu Kapitel 5 „Funktionen“

### 1. Messagebox

Erstelle ein JS, das nach dem Klick auf einen Button eine Funktion `nachricht()` aufruft, in der eine Messagebox mit dem Text „Du hast auf den Button geklickt!“ ausgibt.

Hinweis: Verwende zur Erstellung des Buttons ein HTML-Formular und den Eventhandler `onclick` für den Aufruf der Funktion.

### 2. Altersberechnung

Erstelle ein JS, das aufgrund einer eingegebenen Geburts-Jahreszahl und dem Klick auf einen Button „Alter berechnen“ eine Funktion `alter()` aufruft, in der das Lebensalter berechnet und ausgegeben wird. Prüfe auch die **Plausibilität** des eingegebenen Geburtsjahrs und gib entsprechende Meldungen aus.

### 3. Rechner für die Grundrechnungsarten

Erstelle eine HTML-Datei, in der in einem Formular in 3 Feldern zwei Zahlen und ein Rechenzeichen (+, -, \*, /) eingelesen werden. Nach Druck auf einen Button „Berechnen“ soll eine JS-Funktion `berechnen()` aufgerufen werden, in der die entsprechende Rechnung durchgeführt und das Ergebnis ausgegeben wird. Prüfe auch die **Plausibilität** der eingegebenen Werte und gib entsprechende Meldungen aus.

### 4. Rechner für die Kreisfunktionen

Erstelle eine HTML-Datei, in der in einem Formularfeld der Radius eingelesen werden kann. Nach Druck auf einen Button „Berechne“ bzw. „Fläche“ soll eine JS-Funktion `durchmesser()` bzw. `kreisumfang()` bzw. `kreisflaeche()` aufgerufen werden, in der die entsprechenden Rechnungen durchgeführt und das Ergebnis ausgegeben werden. Prüfe auch die **Plausibilität** der eingegebenen Werte und gib entsprechende Meldungen aus.

Online Rechner: Kreisumfang, Kreisfläche berechnen

Radius:	<input type="text" value="3"/>
Durchmesser:	<input type="text" value="6"/>
Umfang:	<input type="text" value="18.85"/>
Flächeninhalt:	<input type="text" value="28.274"/>
<input type="button" value="Berechne"/> <input type="button" value="Lösche"/>	

Zusatz: Überlege dir eine Möglichkeit Einheiten zu berücksichtigen!

# Kapitel 6 „Objekte“

JavaScript kennt Objekte unterschiedlicher Arten:

- *Eigene bzw. benutzerdefinierte Objekte*: Hierbei handelt es sich um Objekte, die man als Entwickler selbst in JavaScript erstellt. JavaScript bietet dazu verschiedene Techniken an.
- *Vordefinierte Objekte*: Dies sind die nativen Objekte, die von JavaScript als Teil der Sprache zur Verfügung gestellt werden. Zu den von JavaScript bereitgestellten fertigen Objekten gehören `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `RegExp`, `String` und `Object`.
- *Browser-Objekte*: Die Browser-Objekte (auch: Host-Objekte) werden von der Laufzeitumgebung zur Verfügung gestellt. Im Webbrowser stellt zB. das Objekt `window` das Browserfenster dar. Für den Inhalt des Browserfensters steht das `document`-Objekt zur Verfügung, womit man Zugriff auf die HTML-Elemente hat. Weitere nützliche Objekte des Webbrowsers sind `history` und `location`.

## Benutzerdefinierte Objekte in JavaScript

Vereinfacht ausgedrückt, sind Objekte in JavaScript im Grunde nichts anderes als komplexe und zusammengesetzte Variablen mit Eigenschaften und Methoden. Die Eigenschaften eines Objekts werden auch *Attribute* oder *Properties* und die Methoden gelegentlich auch *Objektmethoden* genannt. Auf alle diese Informationen wie Eigenschaften und Methoden hat man Zugriff mit dem Objekt.

Bsp.:

```
var person = {
    vname : "Max",
    nname : "Mustermann",
    alter : 41,
    daten_ausgeben : function() {
        console.log("Vorname : " + this.vname);
        console.log("Nachname : " + this.nname);
        console.log("Alter : " + this.alter);
    },
    kompletter_name : function() {
        return this.vname + " " + this.nname;
    }
}

person.daten_ausgeben();
var name = person.kompletter_name();
console.log("Hallo " + name);
```

Alles, was hier hinter der Zuweisung an `var person` zwischen den geschweiften Klammern steht, ist der Inhalt des Objekts. Im Beispiel sind es drei *Eigenschaften* (die Daten) mit `vname`, `nname` und `alter` und zwei *Methoden* mit `daten_ausgeben` und `kompletter_name`. Methoden werden in JavaScript wiederum mit dem Schlüsselwort `function` eingeleitet. Das Schlüsselwort kann aber auch weggelassen werden. Die einzelnen Eigenschaften und Methoden eines Objekts sind mit *Kommata* zu trennen, und der Wert einer Eigenschaft bzw. Methode wird hinter einem Doppelpunkt notiert.

In diesem Beispiel erhält man nur noch ein globales Objekt `person` im `window`-Objekt für den Zugriff (`window.person`).

Der lesbare Zugriff auf die Eigenschaften eines Objekts kann entweder über die Punkt-Notation (`Objektname.Eigenschaft`) oder die `[]`-Schreibweise (`Objektname["Eigenschaft"]`) erfolgen.

Bsp.:

```
console.log(person.vname);
console.log(person["nname"]);
console.log(person['alter']);
```

Auch der Zugriff zum Ändern der Eigenschaften funktioniert auf diese Weise. Hier das Beispiel, um die Eigenschaften eines Objekts zu verändern:

```
person.vname = "Maxine";
person["nname"] = "Musterfrau";
person['alter'] = 33;
person.daten_ausgeben();
```

Eigenschaften aus dem Objekt entfernen:

```
delete person.alter;           // oder: delete person["alter"];
console.log(person.alter);     // undefined
```

Bsp.: Zugriff auf Methoden

```
person.daten_ausgeben();
console.log("Kompletter Namen: " + person["kompletter_name"]());
person['daten_ausgeben']();
var name = person.kompletter_name();
```

## Möglichkeiten, neue Objekte zu erzeugen

Bsp.: Konstruktorfunktion, aus der man anschließend beliebig viele Objekte erzeugen kann:

```
function Person(vn, nn, a) {
    this.vname = vn;
    this.nname = nn;
    this.alter = a;
    this.print = daten_ausgeben;
}
var daten_ausgeben = function() {
    console.log("Vorname : " + this.vname);
    console.log("Nachname : " + this.nname);
    console.log("Alter : " + this.alter);
}
```

Jetzt kann man mit `new` eine neue Instanz von `Person` erzeugen und die Methode `print` (alias `daten_ausgeben`) im Ausführungskontext des erzeugten Objekts ausführen. Dieses `this` im Beispiel bezieht sich auf die erzeugte Instanz und sorgt dafür, dass die Eigenschaften `vname`, `nname` und `alter` des Objekts auf den Wert des Funktionsparameters `vn`, `nn` und `a` gesetzt werden.

Bsp.: Zwei neue Instanzen von `Person`

```
var kunde01 = new Person("Max", "Mustermann", 41);
var kunde02 = new Person("Maxi", "Musterfrau", 33);
kunde01.print();
kunde02["print"]();
```

## Wiederholungsfragen zu Kapitel 6 „Objekte“

1. Welche Objekte stehen in JS zur Verfügung?
2. Was sind benutzerdefinierte Objekte in JS?
3. Was versteht man unter „Eigenschaften“ eines Objekts?
4. Wie können Objekteigenschaften definiert werden?
5. Was versteht man unter „Methoden“ eines Objekts?
6. Wie können Objektmethoden definiert werden?
7. Was versteht man unter einer Objektinstanz?
8. Wie können neue Instanzen eines Objekts erzeugt werden?
9. Wie können Eigenschaften aus einem Objekt entfernt werden?

## Übungsbeispiele zu Kapitel 6 „Objekte“

### Eigenschaften

1. Raumschiff  
Realisiere das Skript von S. 60 und lege Instanzen für 3 Raumschiffe (Enterprise, Voyager, Challenger) an.
2. Raumschiff – Personen  
Erweitere obiges Skript um ein Objekt Person (siehe S. 62)!  
Lege Instanzen für 3 Personen (Kirk, Janeway, Archer) an!
3. Raumschiff – Kapitän  
Erweitere obiges Skript und weise jedem Raumschiff einen Kapitän zu (S. 63)!

### Methoden

4. Raumschiff – Bewegung  
Realisiere das Skript von S. 65 und lege Instanzen für 3 Raumschiffe (Enterprise, Voyager, Challenger) an.  
Gib für jedes Raumschiff 5 Bewegungsdaten an!
5. Raumschiff – Bewegung 2 (Enterprise)  
Erweitere obiges Skript lt. Angabe S. 68, Punkt 1
6. Raumschiff – Bewegung 3 (Voyager)  
Modifiziere obiges Skript lt. Angabe S. 69, Punkt 1

## Kapitel 7 „Vordefinierte Objekte“

Vordefinierte Objekte sind die nativen Objekte, die von JavaScript als Teil der Sprache zur Verfügung gestellt werden. Zu diesen von JavaScript bereitgestellten fertigen Objekten gehören `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `RegExp`, `String` und `Object`.

### Das oberste Objekt „Object“

Von diesem Objekt stammen alle anderen Objekte in JavaScript ab. Jedes Objekt in JavaScript ist auf jeden Fall vom Typ `Object` und kann darüber hinaus spezifischeren Typen angehören (z. B. `String`). Das `Object`-Objekt stellt Eigenschaften und Methoden bereit, die allen anderen Objekten für die Arbeit damit zur Verfügung stehen. Vom Prinzip her ist ein Objekt vom Typ `Object` nur ein Behälter für Daten wie Zeichenketten oder Zahlen, aber auch Funktionsobjekte lassen sich darin verpacken. Sie können ein Objekt in der Langschreibweise mit `new Object` wie folgt erzeugen:

```
var data = new Object();
data.name = "John Doe";
data.kontonummer = 34234123;
data.blz = 7200032123;
```

Oder du kannst einfach nur wie gehabt als Objekt-Literal wie folgt notieren:

```
var data = {
  name : "John Doe",
  kontonummer : 34234123,
  blz : 7200032123
};
```

### Objekte für die primitiven Datentypen Number, String und Boolean

Die primitiven Datentypen für Zahlen, Strings und Wahrheitswerte hast du bereits regelmäßig verwendet.

Bsp.: `var iVal = 1234;`  
`console.log(typeof iVal); // number`

In JavaScript gibt es für die primitiven Datentypen auch vollwertige Objektversionen.

Bsp.: `var iOVal = new Number(1234);`  
`console.log(typeof iOVal); // object`

In der Praxis wird allerdings empfohlen, nicht die Objektversion der primitiven Datentypen zu verwenden, sondern es bei der primitiven Form zu belassen. Die Objektversion macht den Code nur komplizierter und bremst die Ausführungsgeschwindigkeit des Skripts.

Und wenn du die Methoden von `String`-, `Number`- oder `Boolean`-Objekten verwenden willst, welche zur Verfügung gestellt werden, wird der primitive Datentyp in ein entsprechendes Objekt umgewandelt. Zum Beispiel:

```
var str = "Zeichenkette";
console.log("Anzahl Zeichen: " + str.length); // 12
console.log(str.toUpperCase()); // ZEICHENKETTE

var iVal = 1234;
console.log("1234 als Dualzahl : " + iVal.toString(2)); // 10011010010
console.log("1234 als Hexadezimalzahl : " + iVal.toString(16)); // 4d2
console.log("1234 als Oktalzahl : " + iVal.toString(8)); // 2322
```

## „Math“-Objekt

Für verschiedene Arten von Berechnungen steht das Math-Objekt mit nützlichen Eigenschaften und Methoden zur Verfügung. Du kannst die Eigenschaften und Methoden direkt über `Math.Eigenschaft` bzw. `Math.Methode()` verwenden.

Bsp.:

```
console.log("Konstante für Pi : " + Math.PI); // 3.141592653589793  
var r = 12;  
var a = r * r * Math.PI; // Kreisfläche berechnen  
console.log(a);  
  
console.log(Math.random()); // Pseudo-Zufallszahl zwischen 0 und 1 erzeugen
```

## Reguläre Ausdrücke („RegExp“-Objekte)

Reguläre Ausdrücke dienen dazu, ein Muster von Zeichenketten zu beschreiben, um einen Suchausdruck zu formulieren. Als regulärer Ausdruck wird eine formale Sprache bezeichnet, mit der sich eine (Unter)menge von Zeichenketten beschreiben lässt und die z. B. beim Suchen und/oder Ersetzen verwendet werden kann.

Reguläre Ausdrücke sind Objekte vom Typ `RegExp` und können entweder als Konstruktorfunktion mit `new RegExp()` oder als Literal-Schreibweise erzeugt werden. Hierzu ein einfaches Beispiel:

```
var text = "In diesem Text wird gesucht";  
var regEx01 = /Text/; // Literal-Schreibweise  
var regEx02 = new RegExp(/wird/); // Konstruktorfunktion  
  
var n01 = text.search(regEx01); // Suche nach "Text" in text  
console.log('Text' gefunden an Pos. ' + n01);  
var n02 = text.search(regEx02); // Suche nach "wird" in text  
console.log('wird' gefunden an Pos. ' + n02);  
  
var neuText = text.replace(regEx01, "Absatz"); // Suchen und Ersetzen  
console.log(neuText); // = In diesem Absatz wird gesucht.
```

## „Array“-Objekt

Auch hier kannst du ein neues Objekt über die Konstruktorfunktion `new Array()` oder über die Literal-Kurzschriftweise erzeugen. Mit Arrays kann man gleiche Objekte in einer geordneten Reihenfolge speichern. Hierzu ein Beispiel mit der *Konstruktorfunktion-Schreibweise*:

```
var ort01 = new Array();  
ort01[0] = "Frankfurt";  
ort01[1] = "Stuttgart";  
ort01[2] = "Paris";
```

Dasselbe erreicht man mit der *Literal-Kurzschriftweise*:

```
var ort02 = ["Frankfurt", "Stuttgart", "Paris"];
```

Im Gegensatz zu den primitiven Datentypen wird bei den Arrays sowohl mit der Konstruktorfunktion als auch mit der Literal-Kurzschriftweise ein Objekt erzeugt. Somit sind hier beide Schreibweisen identisch. In der Praxis findet man vorwiegend die Literal-Schreibweise, da diese wesentlich einfacher und besser zu lesen ist.

Beispielsweise erzeugt folgende Schreibweise ein Array mit zwei Elementen:

```
var coordinates = new Array(50, 10); // Array mit zwei Elementen
```

Aber mit folgender Schreibweise erzeugen Sie ein Array mit 50 (!) Elementen:

```
var coordinates = new Array(50) // Array mit 50 undefinierten Elementen
```

Es stehen eine Menge nützliche Methoden und Eigenschaften zur Verfügung. Zum Beispiel:

```
ort01.push("Augsburg");      // Hinten hinzufügen
ort01.push("Zwickau");       // Hinten hinzufügen
ort01.sort();                // Alphabetisch sortieren
ort01.forEach(function(val) { console.log(val) });    // Elemente durchlaufen
console.log("Anzahl Elemente im Array : " + ort01.length);
```

## Wiederholungsfragen zu Kapitel 7 „Vordefinierte Objekte“

1. Welche vordefinierten Objekte stehen in JS zur Verfügung?
2. Warum solltest du für die primitiven Datentypen keine Objektversion verwenden?
3. Durch welche Begrenzer wird in regulären Ausdrücken der Anfang und das Ende des Musters gekennzeichnet?
4. Welche Flags können bei regulären Ausdrücken angegeben werden?
5. Mit welchem Metazeichen kann in regulären Ausdrücken ein Leerzeichen angegeben werden?
6. Was versteht man unter einem Array?
7. Welchen wesentlichen Unterschied gibt es bei der Array-Definition im Gegensatz zu den primitiven Datentypen?
8. In welchen Schreibweisen können Arrays definiert werden?

## Übungsbeispiele zu Kapitel 7 „Vordefinierte Objekte“

### Objekte für die primitiven Datentypen Number, String und Boolean

#### 1. Number-Objekt

Realisiere das nachfolgende Skript mit den Definitionen aus diesem Skriptum. Was fällt auf?

```
if( iVal == iOVal) {
    console.log("Der Wert von iVal ist gleich iOVal.");
}
else {
    console.log("Der Wert von iVal ist nicht gleich iOVal.");
}

if( iVal === iOVal ) {
    console.log("Der Wert und Typ von iVal ist gleich iOVal.");
}
else {
    console.log("Der Wert und Typ von iVal ist nicht gleich iOVal.");
}
```

*Hinweis:* Ein Vergleich mit dem ===-Operator vergleicht auch den Typ.

#### 2. Zeichen zählen

Schreibe ein JS, das die Anzahl eines bestimmten Zeichens in einem String zählt und ausgibt!

## „Math“-Objekt und „RegExp“-Objekte

3. Extrahiere aus folgendem String die Länge und die Breite des Gartens und berechne seine Fläche und Diagonale!

```
var garten = "Laenge=110m, Breite=50m";
```

*Hinweis:* Hilfreich ist das Beispiel im Buch S. 78 und S. 79.

4. Schreibe eine Funktion, die alle Vorkommen der Buchstaben ä, ö, ü und ß in einer eingegebenen Zeichenkette in æ, œ, ue und sz umsetzt und die neue Zeichenkette als Ergebnis zurückliefert und ausgibt:
  - a. Löse das Beispiel mit den Methoden des `string`-Objekts.
  - b. Löse das Beispiel durch die Verwendung von regulären Ausdrücken.

## „Array“-Objekt

5. Entwickle eine Funktion, die den Spruch des Tages zurückliefert. Die einzelnen Texte sollen in einem Array verwaltet werden. Der Spruch des Tages wird über eine Zufallsfunktion ausgewählt.
6. Elemente eines Arrays ausgeben

Erstelle ein Array mit 10 beliebigen Elementen (zB. Automarken). Zeige die Elemente sortiert am Bildschirm an und stelle ein Formularfeld zur Verfügung in dem eine Zahl von 1 bis 10 eingegeben werden kann. Je nach Eingabe soll das entsprechende Element am Bildschirm ausgegeben werden.

7. Formularfelder überprüfen

Erstelle ein Formular mit 5 verschiedenen Typen von Eingabefeldern. Rufe nach Druck auf den Button „Formular absenden“ eine Funktion auf, in der alle Felder auf ihre Eingaben überprüft werden (alle Felder müssen verpflichtend ausgefüllt sein). Eines der Felder soll eine E-Mail-Adresse einlesen, eine entsprechende Überprüfung ob eine E-Mail-Adresse eingegeben wurde soll erfolgen.

E-Mail:

Passwort:

Kommentar:

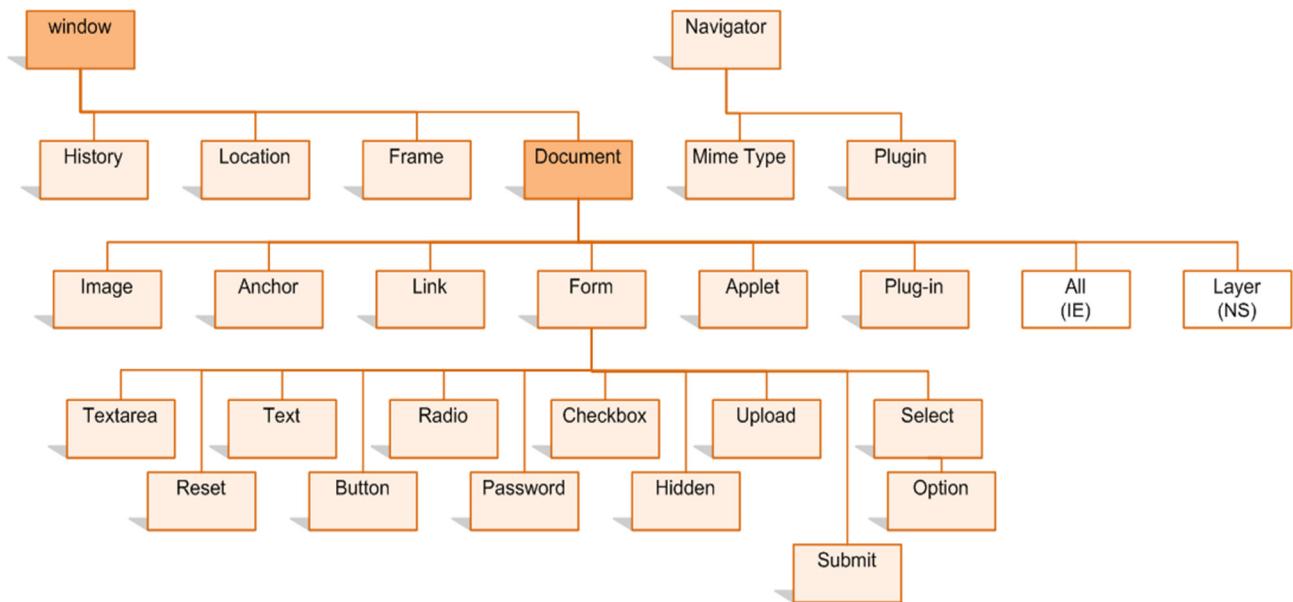
Optionen:  Option 1

Sie stimmen den Lizenzvereinbarungen bei:

Sind alle Eingaben korrekt, so soll der Text „Formular erfolgreich versandt.“ ausgegeben werden.

## Kapitel 8 „DOM-Konzept – Browser-Objekte“

Objekte in JavaScript unterliegen einer **hierarchischen Struktur**. Die verfügbaren Methoden und Eigenschaften dieser Objekte können mit JavaScript abgefragt/verwendet werden. Dabei ist die hierarchische Struktur zu beachten. Begonnen wird mit der obersten Ebene. Die oberste Ebene der Objekthierarchie wird durch das Browserfenster repräsentiert. Die dem Objekt `document` untergeordneten Objekte werden auch dem DOM zugeordnet. Für das `navigator`- und das `screen`-Objekt existieren Hierarchien, die unabhängig vom `window`-Objekt sind.



Mit dem Document Object Model (DOM) ist eine Struktur zum Zugriff auf alle Objekte einer Webseite standardisiert. Beim Laden einer Webseite generiert der Webbrowser eine Baumstruktur der indizierbaren HTML-Elemente. Die Struktur wird erst beim Verlassen der Webseite wieder aus dem Hauptspeicher entfernt. Zwischen den HTML-Elementen bestehen somit Verwandtschaftsbeziehungen. Ein Formular ist Kind-Element von `document`, das Eltern-Element von `formular` und Kind-Element von `window` ist. Der Weg durch den Strukturabaum zum Zugriff auf ein Formular wird beispielsweise derart realisiert:

```
window.document.forms[0];
```

Zugriffsmöglichkeiten auf die Elemente einer Webseite sind gegeben durch

- die Element ID `getElementById()`
- den Namen `getElementsByName()`
- den Elementnamen `getElementsByTagName()`
- über die Objektarrays `window.document.forms[0].elements[1]`

## Wiederholungsfragen zu Kapitel 8 „DOM-Konzept – Browser-Objekte“

1. Was bedeutet die Abkürzung DOM?
2. Wer hat das DOM-Konzept ursprünglich entwickelt?
3. Von wem wird die Objektschnittstelle des DOM heute zur Verfügung gestellt?
4. Wie lauten die hierarchiehöchsten Objekte des DOM?
5. Welche Kindelemente sind dem `document`-Objekt zuordnbar?
6. Welche Zugriffsmöglichkeiten auf die Elemente einer Webseite gibt es?

## Übungsbeispiele zu Kapitel 8 „DOM-Konzept – Browser-Objekte“

### Window-Objekt

1. Erstelle in einer HTML-Seite einen Link auf die Schulwebseite. Die Seite soll mit Hilfe von JS in einem neuen Browserfenster öffnen, das folgende Eigenschaften aufweist: Name des Fensters = Schulwebseite. Die Toolbar und die Adresszeile sollen nicht angezeigt werden. Die Statuszeile ist nicht sichtbar. Die Menüleiste und die Laufleisten werden angezeigt. Das Fenster ist 1024 x 600 Pixel groß und kann größtmäßig nicht verändert werden.
2. Erstelle ein JS, das beim Klick auf einen Button in einem Textfeld nach dem Ablauf von 2 Sekunden den Text „2 Sekunden“ anzeigt. Ebenso sind entsprechende Texte nach 4 Sekunden und nach 6 Sekunden anzuzeigen.

Klicke auf den Button um im Textfeld den Ablauf von 2, 4 bzw. 6 Sekunden anzuzeigen.

3. Erstelle ein JS, das ein Bestätigungsfenster aufruft mit der Frage „Zum ersten Mal hier?“. Je nach Auswahl sind entsprechende Meldungen auszugeben.

### Document-Objekt

4. Erstelle dynamisch eine Webseite, auf der eine Überschrift und ein Bild angezeigt werden. Nutze dazu die Methoden von `document` und `node` (siehe S. 94 ff).

### History-Objekt

5. Schreibe in eine HTML-Seite, die auf eine andere HTML-Seite verlinkt. Von der 2. Seite soll mit einem „Zurück“-Button mit Hilfe des History-Objektes auf die erste Seite zurückmanövriert werden können.

### Location-Objekt

6. Schreibe in eine HTML-Seite, die bei Aufruf automatisch auf die Schulwebseite weiterleitet.

### Screen-Objekt

7. Schreibe ein JS, das die Bildschirmauflösung, den verfügbaren Bildbereich und die Farbtiefe identifiziert und ausgibt. Wenn die Bildschirmauflösung in der Breite kleiner als 1200 ist soll, „Ist wohl ein alter Monitor!“ ausgegeben werden, ansonsten „Schaut nach einem neuwertigen Monitor aus!“

### Navigator-Objekt

8. Schreibe ein JS, das alle wesentlichen Informationen des verwendeten Browsers anzeigt (Name, Version, Plattform, Codename, Cookies, Sprache, Useragent). Überprüfe auch ob JavaScript in den Optionen eingeschaltet ist und gib eine entsprechende Textmeldung am Bildschirm aus.
9. Schreibe ein JS, das den Browser identifiziert und je nach Browser entweder die Seite `ie.html` oder `firefox.html` aufruft. Darin soll eine Textmeldung ausgegeben werden, welcher Browser verwendet wird.

## Kapitel 9 „Ereignisse“

Richtig interaktiv kann man Webseiten mithilfe von JavaScript-Ereignissen (oder auch *JavaScript-Events*) erstellen. Typische Beispiele sind das Wechseln von Bildern, wenn man mit der Maus über einem HTML-Element steht, oder das Überprüfen einer Eingabe bei Formularen.

Das Prinzip ist relativ einfach: Der Webbrower erzeugt ein *Event*, wenn im Dokument, im Webbrower oder bei einem bestimmten HTML-Element eine Aktion ausgeführt wurde. Der Webbrower erzeugt beispielsweise ein Event, wenn die Webseite komplett geladen, die Maus bewegt oder eine Schaltfläche angeklickt wurde. Wenn man sich für ein bestimmtes Event interessiert, kann man eine Handler-Funktion dafür registrieren, welche ausgeführt wird, wenn ein Event von diesem Typ ausgelöst wurde.

JavaScript stellt Ereignisse zu folgenden Bereichen zur Verfügung:

- Ereignisse der Benutzeroberfläche des Fensters (wie `onload`, `onunload`, `onresize`, `onscroll`, `onerror` und `onabort`)
- Maus-Ereignisse (wie `onclick`, `ondblclick`, `onmousedown`, `onmousemove`, `onmouseover`, `onmouseout` und `onmouseup`)
- Tastatur-Ereignisse (wie `onkeypress`, `onkeydown` und `onkeyup`)
- Formular-Ereignisse (wie `onblur`, `onchange`, `onfocus`, `onreset`, `onselect` und `onsubmit`)
- Touch-Ereignisse (wie `touchstart`, `touchend`, `touchcancel`, `touchleave` und `touchmove`)
- Ereignisse für das Abspielen von Video und Audio (wie `onplay`, `oncanplay`, `onpause`, `oncanplaythrough`, `onplaying`, `ondurationchange`, `onvolumechange` und `onended`)
- Drag&Drop-Ereignisse (wie `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart` und `ondrop`)
- Animations-Ereignisse für CSS-Animationen

## Wiederholungsfragen zu Kapitel 9 „Ereignisse“

1. Was versteht man unter einem Event-Handler?
2. Für welche Bereiche stehen unter JS Event-Handler zur Verfügung?
3. Warum sollte man HTML-Eventhandler nicht verwenden?
4. Wie kann auf Fensterereignisse reagiert werden?
5. Wie kann auf Dokumentereignisse reagiert werden?
6. Was passiert, wenn auf Bestandteile des DOM-Baumes zu früh zugegriffen wird?
7. Wie wird der Event-Handler `onload` zum Schutz des DOM-Baumes eingesetzt?
8. Wie kann auf Mausereignisse reagiert werden?
9. Wie kann auf Tastaturereignisse reagiert werden?

## Übungsbeispiele zu Kapitel 9 „Ereignisse“

1. Nachrichten beim Betreten bzw. Verlassen einer Seite

Erstelle ein JS, das beim Betreten einer Seite eine MessageBox mit „Hallo“ ausgibt. Beim Betreten soll die MessageBox mit 3 Sekunden Verzögerung angezeigt werden.

`onload, setTimeout`

2. Tastencode ausgeben

Erstelle ein JS, das beim Drücken einer Taste den entsprechenden Tastaturcode ausgibt.

`onkeypress`

3. Textfeld füllen

Erstelle ein JS, das beim Klick auf einen Button, die entsprechende Ziffer in ein Textfeld schreibt.



4. Bilder tauschen

Erstelle ein JS, das den Bewegungsablauf eines Tennis-Vorhandschlags darstellt (Bilder bitte aus dem Internet suchen, zB <http://www.dr-gumpert.de/html/vorhand.html>). Beim **Berühren der Schaltflächen** „Ausholen“, „Treffen“ und „Ausschwingen“ soll das jeweilige Bild anstelle des ursprünglichen Bildes angezeigt werden.



Ansicht nach dem Laden



Ansicht nach Berühren der Schaltfläche „Ausholen“



Ansicht nach Berühren der Schaltfläche „Treffen“



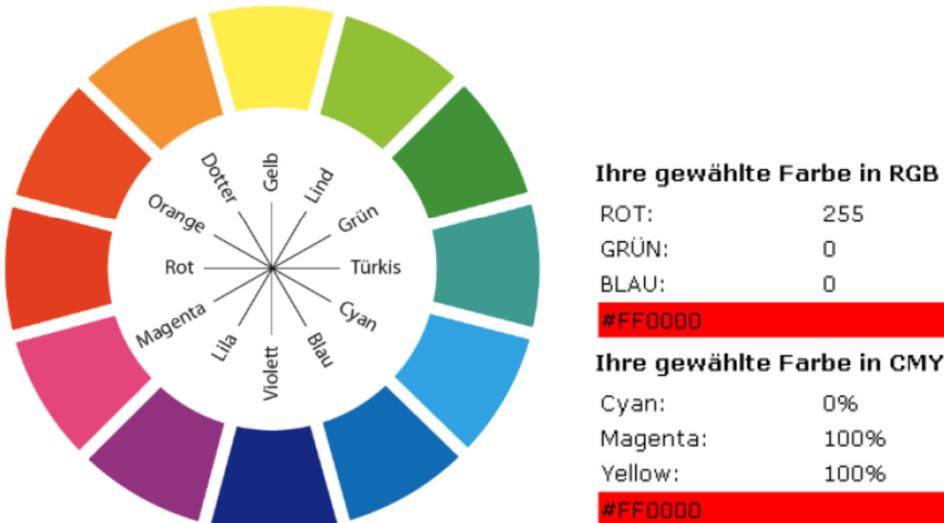
Ansicht nach Berühren der Schaltfläche „Ausschwingen“

## Abschließendes Beispiel

Implementieren ein JavaScript, das im Farbkreis nach Küppers nach Auswahl der Farbe die gewählte Farbe in RGB und CMY anzeigt. Die Webseite soll folgendes Aussehen haben:

Farbkreis nach Küppers

Wählen Sie nachfolgend eine Primär- oder Sekundärfarbe aus dem Farbkreis.  
Dieser Text wird dann ebenso in dieser Farbe automatisch dargestellt.



- Überschrift: "Farbkreis nach Küppers"
- Einführender Text: "Wählen Sie nachfolgend (aus Sicht von RGB) eine Primär- oder Sekundärfarbe aus dem Farbkreis. Dieser Text wird dann ebenso in dieser Farbe automatisch dargestellt."
- Darstellung des Farbkreises nach Küppers (Hinweis: Verwende eine entsprechende Grafik).
- Darstellung der gewählten Farbe in RGB in Text und Farbe.
- Darstellung der gewählten Farbe in CMY in Text und Farbe.

Das Skript soll folgende Anforderungen verwirklichen:

- Der Benutzer kann auf eine Primärfarbe (Rot, Grün, Blau) oder Sekundärfarbe (Cyan, Magenta, Gelb) klicken. Alle anderen Farben müssen auf den Mausklick nicht reagieren.
- Beim Klick des Benutzers auf eine dieser Farben sollen die Werte und der Farbbalken in RGB und CMY entsprechend aktualisiert werden. Die Darstellung als Hex-Wert soll im Farbbalken erscheinen. Im selben Moment soll auch der einführende Text in dieser Farbe dargestellt werden.

Hinweise zur Implementierung:

- Verwende zur Auswahl der Farbe im Farbkreis eine Image Map.
- In der Image Map kannst du via `<area href="javascript:...>` auf den Benutzerklick reagieren und entsprechende JavaScript Funktionen aufrufen.
- Die Funktion `rgb(255, 0, 0)` erlaubt die Definition einer Farbe in RGB.
- Beachte die Komposition (aus Sicht von RGB) der Primärfarben durch die Sekundärfarben, zB. 100% Magenta und 100% Yellow ergeben Rot, dh. stelle die Sekundärfarben mit Hilfe von RGB dar.

Das abschließende Beispiel ist **eigenständig zu lösen**. Kopierte Leistungen werden nicht anerkannt!