Titel: JavaScript. The Definitive Guide
Autor: David Flanagan
Preis: 48,00 EUR
ISBN: 978-0-596-80552-4

1 Allgemeine Bemerkungen

Das Buch richtet sich an Leser mit grundlegenden Kenntnissen in der Programmierung von JavaScript. Auch ohne Grundkenntnisse In JavaScript ist es durchaus möglich (wie hier geschehen), das ganze Buch durchzuarbeiten, aber dies erfordert angesichts des Buchumfanges viel Geduld und Disziplin.

2 Gliederung des Buches

Das Buch ist in vier Teile gegliedert. Das erste Kapitel stellt die Einführung dar, in der unter anderem auf die Entstehung des Namens eingegangen wird. Zu Beginn jedes Kapitels wird eine kurze Übersicht über den kommenden Inhalt geboten. Ausserdem findet man in jedem Kapitel zahlreiche Beispiel-Programme.

2.1 Part I: Core JavaScript

2.1.1 Kapitel 2: Lexical Structure

Im zweiten Kapitel stellt der Autor kurz die allgemeinen Regeln der Verwendung von Groß- und Kleinschreibung in Programmen mit JavaScript vor. Desweiteren erklärt er wie Kommentare markiert werden, was *Literalen* und *Identifiers* sind, und über welche *reservierten Wörter* JavaScript verfügt.

2.1.2 Kapitel 3: Types, Values, and Variables

Im dritten Kapitel gibt der Autor eine Klassifizierung der Variablentypen und erläutert, wie sie ineinander konvertiert werden können. Er unterteilt die Typen in zwei Kategorien: *primitive types*, zu denen Zahlen und Strings gehören, und andererseits *object types*, zu denen Arrays und Funktionen gehören.

Der Autor geht hierbei sehr ausführlich auf die Zahlen und später auf Strings ein.

Im weiteren Verlauf des Kapitels werden *global objects* und *wrapper Objects* behandelt.

Der Autor erklärt den Unterschied zwischen *Immutable Primitive Values* (z.B. Zahlen und Strings) und *Mutable Object References* (z.B. Objekte, wie z.B. Arrays). Im ersteren Falle haben wir es (auch bei Strings!) mit unveränderlichen

Entitäten zu tun, während bei Objekten lediglich Referenzen auf Entitäten erzeugt werden.

Wir erfahren, wie man in JavaSkript Zahlen in Strings und umgekehrt konvertiert. Ausserdem lernen wir, wie man Objekte in Strings konvertiert.

Schließlich werden Variablen vorgestellt, die in *globale* und *lokale* Variablen unterteilt werden. Globale Variablen gelten für das ganze Programm, wohingegen lokale Variablen z.B. nur im lokalen Bereich einer Funktion gelten.

2.1.3 Kapitel 4: Expressions and Operators

Im vierten Kapitel stellt der Autor Ausdrücke und Operatoren vor.

Der Autor unterteilt Ausdrücke in

- **primary expressions**, d.h. Konstanten, oder reservierte Wörter wie *true* oder *sum* (gibt den Wert einer Variablen nach einer Berechnung aus)
- Objekt- und Arrayinitialisatoren
- Funktionsdefinitionsausdrücke

Der Autor geht sehr ausführlich auf die Eigenschaften der Ausdrücke und deren Aufruf ein. Schließlich erfahren wir, wie man ein neues Object mithilfe von Ausdrücken erzeugt.

Mithilfe von Operatoren wird die Kombinationen von Ausdrücken überhaupt erst ermöglicht: z.B. Vergleich zweier Ausdrücke usw. Es folgt eine Tabelle der Operatoren.

Es folgt die Erklärung der Begriffe *Lvalue*, *Side Effects*, *Operator Precedence* und *Operator Associativity*.

Es werden folgende Arten von Operatoren behandelt:

- mathematische Operatoren
- bitwise operators, meistens bei Zahlen verwendet;
- Equality and Inequality Operators
- Comparison Operators
- der in-Operator
- der instanceof-Operator

- logische Operatoren: *Logical AND* (&&), *Logical OR* (||) und *Logical NOT* (!)

Ausdrücke mit Zuweisungen können kombinierte Operatoren enthalten, die aus anderen Programmiersprachen bekannt sind, z.B. +=.

Weiter im Kapitel behandelt der Autor Ausdrucksauswertungen (*Evaluation Expressions*) und wie sie in JavaScript funktionieren. Dabei erklärt er folgende Funktionen *eval()*, *Global eval()* und *Strict eval()*.

Schließlich erwähnt er noch folgende Operatoren:

- **Conditional** Operator (?:)
- **typeof** Operator
- **delete** Operator
- **void** Operator
- **Comma** Operator (,)

2.1.4 Kapitel 5: Statements

Im fünften Kapitel werden Anweisungen als Kommandos in Form von Ausdrücken oder Deklarationen vorgestellt, die ausgeführt werden.

Der Autor erklärt die Unterschiede zwischen Ausdrucks- und Deklarationsanweisungen. Die einfachste Art von Anweisungen stellen die Ausdrucks-Anweisungen dar. Zu den Bestandteilen einer Deklarationsanweisung gehören entweder *var*, das eine Variable, oder *function*, das eine Funktion definiert.

Der Autor stellt danach folgende *Konditional*-Anweisungen vor:

- **if**
- **else if**
- **switch**

Anschließend werden Schleifen vorgestellt:

- **while**
- **do/while**
- **for**
- **for/in**

Ausserdem erfahren wir, durch welche Anweisungen *Jumps* in Schleifen dargestellt werden:

- **break**
- **continue**
- **return**
- **throw**
- **try/catch/finally**

(Die letzten zwei Anweisungen werden bei der Fehlerbehandlung verwendet.)

Schließlich werden unter der Rubrik *Miscellaneous Statements* folgende Anweisungen vorgestellt:

- **with**, welches inzwischen als veraltet angesehen wird.
- **debugger**
- **use strict**-Direktive, um in den strikten Modus zuschalten.

2.1.5 Kapitel 6: Objects

Im sechsten Kapitel stellt der Autor den wichtigsten Datentyp von JavaScript vor: *Objekte* und ihre Eigenschaften.

Objekte in JavaScript sind dynamischer Natur, d.h. ihr können Eigenschaften zugewiesen und entnommen werden. Ausserdem sind Objekte austauschbar und werden daher eher durch Referenz verändert als durch Wertzuweisung.

Der Autor erklärt wie Objekte erzeugt werden:

- **Literale**
- **new-Operator**
- **Object.create()-Methode**

Anschliessend erklärt der Autor, wie Objekte erzeugt werden, die von einem Prototypen Eigenschften mitvererbt bekommen.

Der Autor thematisiert sehr ausführlich das *Querying* und *Property Setting* bei Objekten, insbesondere bei den sog. *Objects As Associative Arrays*.

Ausserdem erfahren wir, wie man die Eigenschaften der Objekte löschen, testen und nummerieren kann.

Schließlich behandelt der Autor Eigenschafts- und Objektattribute mit kleinen Beispielcodes in getrenten Unterkapiteln. Bei den Eigenschaftsattributen geht der Autor auf die Prototypen und auf Klassen ein.

Bei Objektattributen geht der Autor kurz auf die Frage der Vereerbung der Attribute ein und verrät uns, wie man Atributtsklassen ermitteln kann.

Dann erfahren wir, welche Erweiterungen bei den Attributen angewendet werden können, z.B.:

Object.seal() um Eigenschaften der Objekte vom Anfügen, Löschen oder Konfigurieren zu schützen.

Object.freeze Eigenschaften der Objekte könen nur gelesen werden.

Object.preventExtensions() Eigenschaften der Objekte werden vor Eingriffen geschützt.

Die Serialisierung der Objekte wird nur kurz angerissen.

Folgende Methoden von Objekten werden vorgestellt:

- **toString()**
- **toLocaleString()**
- **toJSON()**
- **valueOf()**

2.1.6 Kapitel 7: Arrays

Im siebten Kapitel werden Arrays als Objekte behandelt, die dynamischer Natur sind und eine Länge als Eigenschaft haben.

Es wird erklärt, wie Arrays erzeugt werden, wie aus ihnen gelesen wird und wie Elemente in darin geschrieben werden.

Ausserdem wird der Begriff *Sparse Arrays* erklärt und an kurzen Beispielen vorgestellt. Der Begriff Länge ist mit dem Begriff *Sparse* eng verbunden, da es sich dabei um eine Diskrepanz handelt, bei der der Wert der Eigenschaftslänge größer ist als die Menge der Elemente des Arrays. Dies ist z.B. der Fall, wenn man einen *delete*-Operator anwendet und dadurch dieses Arrayelement auf *undefined* gesetzt wird. Zwar ist ein solches Element im Array nicht mehr auffindbar, jedoch bleibt die ursprüngliche Länge des Arrays bestehen.

Es folgen Erklärungen für *push* und *delete* Methoden.

Anschließend stellt der Autor Iterationsmöglichkeiten mithilfe von *for*-Schleife vor. Ausserdem stellt er eine neue ECMAScript 5 Methode vor (*forEach()*).

Weiter im Kapitel erfahren wir, wie man multidimensionale Arrays verwenden kann, z.B. um eine Tabelle darzustellen.

Im Unterkapitel **7.8** stellt der Autor die Array-Methoden im Einzelnen vor:

- **join()**
- **reverse()**
- **sort()**
- **concat()**
- **slice()**
- **splice()**
- **push()** und **pop()**
- **unshift()** und **shift()**
- **toString()** und **toLocaleString()**

Der Autor stellt gesondert davon die in ECMAScript 5 neu definierten Iterierungs-Methoden für Arrays vor:

- **forEach()**
- **map()**
- **filter()**
- **every()** und **some()**
- **reduce()** und **reduceRight()**
- **indexOf()** und **lastIndexOf()**

Danach werden Array-Typen vorgestellt und schließlich erfahren wir, was *Array-like Objects* sind, sowie wann sich *Strings As Arrays* verhalten und wozu dies gut ist.

2.1.7 Kapitel 8: Functions

In diesem Kapitel geht der Autor zunächst sehr ausführlich auf Funktionen in JavaScript ein und bespricht deren Bestandteile, Aufbau und natürlich nicht zuletzt ihre Hauptaufgabe – etwas ausführen.

Anschließend wird vorgeführt, wie eine Funktion definiert wird. Dabei wird der Begriff *nested functions* erklärt (es handelt sich hierbei um eine in einer anderen Funktion eingebettete Funktion).

Nachdem eine Funktion definiert wurde, kann sie in JavaScript auf vier Weisen aufgerufen werden:

as functions eine Standardfunktion, wie in anderen Sprachen.

as methods d.h. wenn eine JavaScript-Funktion in der Eigenschaft eines Objekts oder als Element eines Arrays gespeichert ist (engl. *stored*).

as constructors d.h. wenn eine Funktion oder eine Methode mit dem Schlüssel **new** gefolgt von einem Objekt beginnt. Hierbei unterscheidet sich der Konstruktor von den beiden oben erwähnten Arten des Funktionsaufrufes dadurch, dass die runden Klammern für die Parameter weggelassen werden können. Beim Aufruf des Konstruktors wird ein leeres Objekt erzeugt, welches prototypische Eigenschaften des Konstruktors erbt und auf welches der Konstruktor mit dem Schlüssel **this** referieren kann. Die Konstruktoren haben hier jedoch keinen *return*-Schlüssel, wie es z.B. bei gewöhnlichen Funktionen der Fall ist. Das neue Objekt ist stattdessen der Wert des Konstruktor-Aufrufs.

indirect invocation durch die Methoden *call()* und *apply()*. Diese beiden Methoden erlauben den **this**-Wert für den Funktionsaufruf explizit zu bestimmen. Dadurch wird es möglich, jede Funktion und jede Methode jeden Objektes aufzurufen, auch wenn es sich nicht um eine Methode dieses Objektes handelt. Mit diesen beiden Methoden kann man ebenso Argumente für den Funktionsaufruf bestimmen: *call()* verwendet dabei eine eigene Liste und *apply()* erwartet ein Array von Werten, die als Argumente verwendet werden.

Im weiteren Verlauf des Kapitels erfahren wir, was passiert, wenn eine Funktion mit weniger Argumenten aufgerufen wird als sie via Parameter deklariert wurde. Der Autor stellt ein Verfahren vor, mithilfe dessen man *undefined*-Werten vorbeugen kann, nämlich indem man im Code von Anfang an solch einen Fall mitberücksichtigt und optionale Parameter mitaufführt.

Eine weitere Lösung sieht der Autor in der Kontrolle der Argumentenliste, und zwar indem man die Argumentenliste mithilfe von *arguments.length* daraufhin überprüft, wieviele Argumente das Objekt tatsächlich hat.

Wir erfahren ausserdem, wie man Funktionen als *Namespaces* deklariert, wenn man eigene Module entwickelt hat und befürchtet, dass es beim Einsatz im Web eventuell Konflikte mit anderen Programmen geben könnte.

Eines der wichtigsten Themen in JavaScript stellen die *Closures* dar. Der Begriff von Closures (dt. Verschluss) beruht auf einem Konzept, das besagt, dass die lokalen Variablen durch einen Geltungsbereich, genauer eine *scope chain*, an die Funktion gebunden sind.

Die Closures werden bevorzugt in Anwendungen mit verschachtelten Funktionen verwendet, da sie dort in einem „anderem“ *scope chain* aufgerufen werden, als in dem, in dem sie ursprünglich definiert wurden. Die Bindung allerdings

besteht weiterhin und kann aus jeder beliebigen Stelle im Code heraus erfolgen.

Der Autor weist daraufhin, dass Closures nicht auf **this**- Schlüssel der äußeren Funktion zugreifen können. Außerdem wird den Closures der Zugang zu den **arguments** der äußeren Funktion verwehrt, die automatisch bei jedem Funktionsaufruf deklariert werden. Eine Lösung für dieses Problem sieht der Autor darin, dass die äußere Funktion die jeweiligen Werte in Variablen speichert.

Als nächstes stellt der Autor Funktionseigenschaften wie *length* vor.

Folgende Methoden werden im weiteren Verlauf erneut jedoch ausführlicher vorgestellt:

- **call()**
- **apply()**
- **bind()**
- **toString()**

Der Autor erklärt, wie ein *Function() Constructor* definiert wird und zählt die wichtigsten Punkte auf, die bei Funktionskonstruktoren zu beachten sind:

- Funktionen werden dabei dynamisch erzeugt und zur Laufzeit kompiliert
- der Funktionskörper wird geparst und es wird bei jedem Aufruf ein neues Funktionsobjekt erzeugt.
- Funktionen, die durch einen Funktionskonstruktor erzeugt werden, verwenden keine lexikalischen Geltungsbereiche, sondern sie werden behandelt wie top-level Funktionen.

Der Autor rät von der Verwendung von *Function() Constructors* ab, da ein Funktionskonstruktor neue Variablen und Funktionen in seinem privaten Geltungsbereich definiert.

Der Autor stellt als nächstes sog. *callable objects* vor. Diese sind Objekte, obwohl sie keine Funktionen sind. Zu ihnen gehören solche Methoden wie z.B. *Window.alert()* und *Document.getElementById()*, welche *callable host objects* verwenden statt der echten Funktionen.

Zu den callable objects gehören auch die *RegExp objects*, die mihilfe der *exec()*-Methode erzeugt werden.

Zum Schluß beschreibt der Autor JavaScript im Licht der funktionalen Programmierung, und führt hierfür zahlreiche Programmcode-Beispiele an.

2.1.8 Kapitel 9: Classes and Modules

In diesem Kapitel stellt der Autor zuerst JavaScript-Klassen vor – wie sie aufgebaut sind, sowie deren syntaktisches Verhalten. Später geht der Autor noch auf Module ein, mit deren Hilfe die Klassen für die künftige Wiederverwendbarkeit refakturiert werden können.

Classes Folgende Begriffe werden im Zusammenhang mit JavaScript-Klassen eingeführt:

Prototype Bei JavaScript Klassen-ist der Begriff des *Prototype* von besonderer Bedeutung, da eine Klasse ein Set von Objekten darstellt und eine Klasse ihre Eigenschaften weitervererbt, die vom gleichen Prototyp sein müssen.

Constructors Konstruktoren erzeugen ein neues Objekt, und zwar mit Hilfe des Schlüssels **new**. Der Prototyp des Kosnstruktors übernimmt die Rolle des Prototyps des neuen Objekts. Alle Objekte, die von demselben Konstruktor erzeugt wurden, erben die gleichen Eigenschften von demselben Objekt und gehören damit zu den Mitgliedern einer Klasse.

Augmenting Classes Klassen kann man dynamisch erweitern, indem man neue Methoden zu den Prototyp-Objekten hinzufügt.

Classes and Types Mit Hilfe des *typeof*-Operators kann man den Typ einer Klasse herausfinden. Mit Hilfe des *instanceof*-Operators kann man ermitteln, von welchem Objekt ein Element abstammt.

Duck-Typing Wenn ein Objekt ein bestimmtes Verhalten aufweist, welches auf eine bestimmte Klasse hinweist, wird es wie diese Klasse behandelt, wenn es auch keine Eigenschaften der Prototyp-Objekte dieser Klasse erbt.

Der Autor stellt objektorientierte Techniken in JavaScript anhand von konkreten Beispielen vor, und erläutert dabei z.B. den *Set()*-Konstruktor und die *enumeration()*-Funktion, mit deren Hilfe sog. *enumerated types* erzeugt werden.

Wir erfahren schließlich ausführlich alles wesentliche über die Standardmethoden zur Umwandlung von Objekten:

toString() mit deren Hilfe die Objekte in Strings umgewandelt werden.

toLocalString() ähnlich wie die oben erwähnte Methode, konvertiert aber auf lokal-sensitive Art. Jedes Objekt erbt diese Methode standardmäßig, wobei es wiederum die *toString()*-Methode aufruft.

valueOf() konvertiert Objekte in primitive Werte. Diese Methode wird automatisch aufgerufen, wenn ein Objekt in einem numerischen Kontext verwendet wird.

toJSON() wird mit Hilfe von *JSON.stringify()* aufgerufen und ist für die Serialisierung von Datenstrukturen bestimmt, wie z.B. primitive Werte, Arrays und einfache Objekte (Eigenschaften wie der Prototyp aber auch der Konstruktor werden dabei ignoriert).

Der Autor stellt als nächstes zwei Vergleichsmethoden vor: *equals()* und *compareTo()*.

Wir erfahren weiterhin, wie man in JavaScript den *private state* eines Objekts erreichen kann.

Schließlich verrät uns der Autor, wie man mehrmalige Initialisierung von Objekten durch Überladung des Konstruktors (*Constructor Overloding*) erreichen kann. Als Alternative wird eine sog. *factory*-Methode an einem Beispiel erörtert.

Der Autor stellt im weiteren Verlauf des Kapitels die Techniken und das dazugehörige Instrumentarium vor, um Subklassen zu erstellen. Wir erfahren dabei ebenfalls, welche Konstruktoren und Methoden explizit bei Subklassen angewandt werden müssen.

Der Autor beschreibt am Ende des Kapitels diejenigen Änderungen in ECMAScript5, die Klassen und Subklassen betreffen.

Im Abschnitt **Module** erklärt der Autor, wie man mit bestimmten Hilfsmitteln Module vor Kollabierung schützen kann. Dazu gehört z.B. das Einrichten von *Namespaces*.

2.1.9 Kapitel 10: Pattern Matching with Regular Expressions

Im zehnten Kapitel stellt der Autor das Verfahren der Regulären Ausdrücke in JavaScript vor, welche aus der Programmiersprache Perl übernommen wurden.

2.1.10 Kapitel 11: JavaScript Subsets and Extensions

Im elften Kapitel werden Untermengen und Erweiterungen von Javascript vorgestellt. Zu den Untergruppen von JavaScript gehören z.B. die sicherheitsrelevanten Codeblöcke. Bei den Erweiterungen ist zu beachten, dass nur diejenigen Erweiterungen im Buch berücksichtigt werden, die unter *Spidermonkey* oder *Rhino*-Interpretern laufen.

Der Autor rät von der Benutzung folgender Sprachelemente ab, da diese der Sicherheit des Codes eher schaden:

- **eval()**-Funktion

- **Function()**-Konstruktor
- **this**-Schlüsselwort
- **with**-Anweisung

Der Autor stellt einige außerdem wertvolle Tools vor, die der Sicherheit der Applikation dienen können:

- **ADsafe**
- **dojox.secure**
- **Caja**
- **FBJS**

Folgende wichtige JavaScript-Erweiterungen werden vorgestellt:

- **destructuring assignement**
- **Iteration** mit Hilfe von *for/each*-Schleife, *next ()*-Methode
- **Generatoren** wie *yield*-Schlüssel.
- **FBJS**

2.1.11 Kapitel 12: Server-Side JavaScript

Im zwölften Kapitel werden zuerst zwei serverseitige JavaScript Interpreter anhand von Beispielen vorgestellt:

- das java-basierte *Rhino*, welches JavaScript Programmen den Zugang zu der gesamten Java API ermöglicht,
- das C++-basierte *Node* welches ansynchrone Zugriffe ermöglicht.

2.2 Part II: Client-Side JavaScript

2.2.1 Kapitel 13: JavaScript in Web Browsers

Im dreizenten Kapitel stellt der Autor die wichtigsten Begriffe des Client-Side-JavaScript vor. Als erstes wird dabei das *Window*-Objekt als Hauptzugangspunkt genannt. Es repräsentiert das Webbrowser-Fenster oder den Frame.

Der Autor erklärt, dass JavaScript bei Web-Dokumenten, die statischer Natur sind anderes eingesetzt wird als bei Web-Applikationen, die zudem die Webbrowser-Umgebung berücksichtigen müssen.

Wir erfahren, wie JavaScript in HTML-Dokumente eingebettet werden kann:

- als Inline, zwischen <script> und </script>
- aus einem externen File mit Hilfe des **src**-Attributes und des <script>-Tags
- mit Hilfe von HTML-Event Handler Attributen, z.B. **onclick** oder **onmousover** u.a.
- in einer URL, welche ein spezielles **javascript: protocol** verwendet.

Der Autor beschreibt anschließend, wie diese auf unterschiedliche Art und Weise eingebetteten JavaScriptcodes dann ausgeführt werden – auch in Bezug auf das *Document Object*. Sie alle teilen nämlich (mit einigen angeführten Ausnahmen) das gleiche *Document Object* und das gleiche Set von globalen Funktionen and Variablen: wenn das Script eine neue Variable oder eine neue Funktion definiert, dann ist diese Variable oder Funktion für jeden anderen JavaScript-Code im Document Object sichtbar und kann bei Bedarf aufgerufen werden nachdem dieser Code ausgeführt wird.

Die Ausführung des Codes verläuft in zwei Phasen:

Phase 1: Script-Ausführung In der ersten Phase wird der Inhalt des Dokuments geladen und der Code (sowohl interner als auch externer Scripte) wird ausgeführt. Die Reihenfolge der Ausführung spiegelt die Eingabereihenfolge wider: von oben nach unten.

Phase 2: Ausführung ansynchroner Ereignisse & Ereignissteuerung In der zweiten Phase werden Event Handler (dazu gehören aber auch **javascript**:URLs) asynchron ausgeführt, welche vorher durch User Input oder durch andere Ereignisse (z.B. ausgelöst durch Netz-Aktivität u.a.) aufgerufen wurden. Hierbei wird auch die Ereignissteuerung näher erklärt, und zwar, wie sie greift, nachdem das Dokument geladen worden ist.

Bei Dokumenten mit **src**-Attribut lädt und führt der Browser zuerst die Skripte mit diesem Attribut aus und blockiert währenddessen das Laden der anderen Teile des Dokumentes. Der Autor stellt in diesem Zusammenhang zwei wichtigen Attribute vor, mit deren Hilfe man dieses Problem umgehen kann:

defer - verzögert das Laden des Skripts, bis das Dokument vollständig geladen ist. Wird ausgeführt wie angegeben worden ist.

async - wird so früh wie möglich ausgefürt, ohne das Laden des Dokumentes zu blockieren. Dieses Attribut wird *defer* bevorzugt, d.h. wenn beide vorhanden sind, wird *async* ausgeführt und *defer* ignoriert. Wird ausgefürt wie es geladen wird.

Anschließend beschreibt der Autor genauen zeitlichen Ablauf der Ausführung des JavaScript-Programms.

Der Autor stellt verschiedene Browser vor und erklärt wie man sie testet. Wir erfahren, wie man den Internet Explorer auskommentiert, wenn es sich um ältere Versionen handelt.

Der Autor führt zudem einen wichtigen Begriff ein: *The Same-Origin Policy.* Es handelt sich dabei um die Handhabung von JavaScript Code von anderen *window*- oder *frame*-Objekten.

2.2.2 Kapitel 14: The Window Object

Im vierzenten Kapitel werden die wichtigsten Eigenschaften und Methoden des *Window*-Objektes vorgestellt:

setTimeout(), setInterval() - mit diesen beiden Window-Methoden werden Funktionen registriert und nach einer festgesetzten Zeit aufgerufen.

location - diese Window-Eigenschaft erlaubt es, die URL des aktuell angezeigten Dokumentes zu ermitteln und ein neues Dokument zu laden.

history - diese Window-Eigenschaft erlaubt es, den Browser in der *history* vorwärts und rückwärts zu bewegen.

navigator - diese Window-Eigenschaft erlaubt es, den Namen und die Version des Browsers zu ermitteln.

screen - diese Window-Eigenschaft erlaubt es, die Größe des Desktops zu ermitteln.

alert(), confirm(), prompt() - Methoden, mit deren Hilfe, einfache Texte angezeigt werden.

showModalDialog() - Methode, mit deren Hilfe eine Dialogbox angezeigt wird.

onerror - Handler-Methode, die aufgerufen wird, wenn eine Fehlermeldung aktiviert wird.

Ausserdem erklärt der Autor, wie die IDs und Namen von HTML-Elementen als Eigenschaften von Window-Objekten verwendet werden.
Und schliesslich erfahren wir, wie man Browser-Fenster öffnet, schließt und wie man JavaScript-Code so schreibt, dass er auf anderen Browser-Fenstern und *nested frames* aktiviert werden kann.

2.2.3 Kapitel 15: Scripting Documents

Im fünfzehnten Kapitel stellt der Autor Techniken vor, wie man mit Hilfe von JavaScripten die Dokument-Objekte des Window-Objekts erzeugen und auf sie zugreifen kann. Dabei spielt das Konzept des *Document Object Models* (*DOM*)

eine zentrale Rolle.

Wir lernen, wie man Elemente des Dokumentes selektiert: durch *ID*, mit Hilfe von Elementen-Namen, durch Element-Typen, durch CSS-Klassen, durch CSS-Selektoren oder per Selektion aller Elemente eines Dokumentes.

Der Autor beschreibt die Dokumentstruktur als Baum und stellt die Funktion der Durchquerung dieses Baums vor. Weiter stellt er die wichtigsten Attribute eines Dokumentes vor: HTML- und Dataset-Attribute.

Der Inhalt eines Dokumentes kann auf drei Arten präsentiert werden:

- als HTML-string, eingeschlossen in HTML-Tags
- als plain-Text, dargestellt in Anführungszeichen
- als Text-Knoten

Es wird ein Beispiel für das Erstellen einer Tabelle vorgestellt.

Der Autor stellt Techniken vor, wie man mit Hilfe von CSS geometrische Daten eines Elementes definieren und sie dynamisch positionieren kann.

Anschließend werden die JavaScript-Techniken vorgestellt, mit deren Hilfe die HTML-Formulare erstellt werden.

Schließlich werden folgende Methoden und Eigenschaften vorgestellt und erklärt:

- document.write()
- getSelection()
- contenteditable
- designMode

2.2.4 Kapitel 16: Scripting CSS

Im sechszehnten Kapitel stellt der Autor Techniken vor, wie man mit JavaScript CSS-Stile nachskripten kann.

2.2.5 Kapitel 17: Handling Events

Im siebzehnten Kapitel geht der Autor sehr ausführlich auf die Darstellung von Handling-Events ein.

Wir erfahren die wichtigsten Begriffe, die konzeptuell mit Handling-Events zusammenhängen und beim Scripting zu beachten sind:

event type wie z.B. Mouse events, Key events u.a.

event target bezeichnet ein Objekt, auf welches sich das Event auswirkt, z.B. ein Button.

event handler, event listener für die Behandlung und das Reagieren auf ein Event.

event object bezeichnet ein Objekt, welches einem bestimmten Event zugeordnet ist und Details über dieses Event beinhaltet.

event propagation ist ein Prozess, bei welchem der Browser entscheidet, bei welchem Objekt ein Event-Handler ausgelöst wird.

event default actions

Der Autor stellt vor, wie man die Events aufruft.

2.2.6 Kapitel 18: Scripted HTTP

Im achtzehnten Kapitel wird dargestellt, wie mit Hilfe von JavaScript eine Kommunikation mit dem Web-Server aufgebaut wird, ohne dass der Inhalt jedes Browser-Fensters oder Frames neu geladen werden muss. Der Autor erklärt dabei die **XMLHttpRequest**-Klasse.

2.2.7 Kapitel 19: The jQuery Library

Im neunzehnten Kapitel stellt der Autor die beliebteste JavaScript-Klassenbibliothek vor, mit deren Hilfe nicht nur DOM-Objekte manipuliert werden können, sondern mit der auch die lästigen Browser-Inkompatibilitäten umschifft werden.

jQuery kann bei Bedarf die Funktionalitäten von *Ajax* (Asynchronous JavaScript and XML) zugreifen, welches bei HTTP-Skripting für Datenübertragung verwendet wird, ohne dabei Seite jeweils neu zu laden. Dieses Verhalten wird mit der Funktion **jQuery.ajax()** erzielt. Der Autor stellt in diesem Zusammenhang weitere Methoden vor:

- *load()*
- *jQuery.getScript()*
- *jQuery.getJSON()*
- *jQuery.get()*
- *jQuery.post()*

Der Autor stellt jQuery-Selektoren und deren Methoden vor.

Wir erfahren das Wichtigste über jQuery-Plugins.

2.2.8 Kapitel 20: Client-Side Storage

Im zwangzigsten Kapitel stellt der Autor Methoden der Client-seitigen Datenspeicherung vor, mit deren Hilfe Daten auf dem Client-Rechner lokal gespeichert werden können.

Der Autor stellt zwei Window-Objekt-Eigenschaften vor, die in diesem Zusammenhang eine wichtige Rolle spielen:

localStorage Diese Eigenschaft verweist auf das Speicherobjekt. Daten, die mit dieser Methode gespeichert worden sind, sind persistent und existieren bis die Web App gelöscht wird oder der Benutzer den Browser veranlasst, sie zu löschen. Der Geltungsbereich ist beschränkt auf das Originaldokument.

sessionStorage Diese Eigenschaft verweist ebenfalls auf das Speicherobjekt. Der Geltungsbereich ist beschränkt auf das Originaldokument. Der Unterschied zu *localStorage* besteht im Bezug auf die Lebensdauer: in diesem Fall ist die Lebensdauer mit der Lebensdauer des Top-Level-Fensters oder des Browser-Tabs identisch, in dem das Script läuft. Wenn das Top-Level-Fenster oder der Browser-Tab permanent geschlossen ist, dann werden die darin gespeicherten Daten durch *sessionStorage* gelöscht. (eine Ausnahme bilden moderne Browser, die die Möglichkeit zum Wiedereröffnen kürzlich geschlossener Tabs bieten).

Der Autor erklärt, wie diese beiden Speichervorgänge realisiert werden und die damit vebundenen *Storage Events*. Ausserdem erfahren wir alles notwendige über *Cookies* und wie man sie mit hilfe von JavaScript realisiert.

Noch eine andere Art der Datenspeicherung wird vorgestellt: *userData*, die es erlaubt, mehr Daten zu speichern als Cookies aber weniger als localStorage oder sessionStorage.

Zum Schluß stellt der Autor den sog. *Application Storage* oder *Application cache* vor und versieht die Darstellung mit einigen Beispielen.

2.2.9 Kapitel 21: Scripted Media and Graphics

Im einundzwangzigsten Kapitel wird vorgestellt, wie man mit Hilfe von JavaScript Bilder und Grafiken manipulieren und erstellen kann. Ausserdem erfahren wir, wie man eine grafische Arbeitsfläche (*Canvas*) definiert.

Es wird erklärt, mit welchen JavaScript-Methoden man Bilder, Grafiken und Audio- und Video-files in das Dokument einbinden kann.

Ausserdem geht der Autor kurz auf die Darstellung von Schriften innerhalb von Grafiken ein.

2.2.10 Kapitel 22: HTML5 APIs

Im zweiundzwangzigsten Kapitel stellt der Autor einige neu APIs aus HTML5 vor, die JavaScript mitimplementieren kann, die aber noch nicht von allen Browsern unterstützt werden. Die Interesantesten unter ihnen sind:

Geolocation API die allen Browsern (nach entsprechender Erlaubnis) ermöglicht, den User geografisch zu lokalisieren.

history management APIs die den Web-Anwendungen erlaubt, den *state in response* zu den Vorwärts- und Rückwärts-Buttons des Browsers zu aktualisieren und zu speichern, ohne dass dabei ein erneutes Laden vom Server erfolgen muss.

Web Workers die JavaScript eine parallele Ausführung von Threads erlauben.

Der Autor stellt ausserdem die JavaScript Standard-Datenbank *IndexedDB* und zum Schluß das *WebSocket Protocol* vor.

2.3 Core JavaScript Reference

Im dritten Teil des Buches werden die wichtigsten Klassen, Methoden und Eigenschaften der *Core*-JavaScript-Sprache alphabetisch aufgelistet. Der Autor verweist gelegentlich auf die jeweilige Kapitel, was sehr nützlich ist. Ebenfalls führt der Autor Beispiele auf.

2.4 Client-Side JavaScript Reference

Im vierten Teil des Buches werden die wichtigsten Client-Side-JavaScript Objekte mit allen Eigenschaften und Methoden in alphabetischer Reihenfolge dargestellt:

- **Window**
- **Document**
- **Element**
- **Event**
- **XMLHttpRequest**
- **Storage**
- **Canvas**
- **File**

Es gibt ausserdem auch einen Eintrag zur *jQuery*-Bibliothek.

3 Kritik

3.1 Tippfehler

Seite 51 zweiter Absatz, vorletzter Satz *An array with a single element converts to the same string that that one element does.*
muss geändert werden in:
An array with a single element converts to the same string that one element does.. (doppeltes *that*)

Seite 139: 6.10.3 The toJSON() Method *See Date.toJSON() for an exaple.*
Hier fehlt die Seitenagabe zur besseren Orientierung: *see Page 762.*

Seite 175: Beispiel, erster Kommentar *Loop though all elements*
muss geändert werden in *through.*

Seite 337: dritter Absatz von oben, zweiter Satz *In this case, the injected script simply displays a dialog box, which is relatively benign.*
muss geändert werden in *begin.*

Seite 350: Beispiel *14-4* das abschließende Kommentarzeichen –> muss umgesetzt werden, da es direkt im Code plaziert wurde.
Muss nach dem Satz *Use this file with code like this:* gesetzt werden.

Seite 381: Beispiel *Recursively convert all Text node descendants of n to uppercase: n.nodeTyep == 4*
muss geändert werden in *n.nodeType.*

Seite 473: vorletzte Zeile *(framwidth, contentwidth)*
muss geändert werden in *framewidth.*

4 Fazit:

Das Buch „JavaScript. The Definitive Guide“ von David Flanagan ist besonders für Web-Programmierer geeignet. Ebenso ist dieses Werk prinzipiell auch für fortgeschrittene Einsteiger in die Programmierung geeignet, die am Beispiel von JavaScript eine wirklich umfassende Einführung in die umfangreiche Welt der aktuellen Fragestellungen in der Programmierung suchen. Man kann mit Fug und Recht behaupten, dass der Autor versucht hat, die komplette Terminologie der aktuellen Programmiertechniken darzustellen. Dazu dient insbesondere die erste Hälfte des Buches mit dem Zwischentitel *Core JavaScript*.

Der eigentliche Hauptanwendungsbereich von JavaScript, nämlich die Erstellung von Web-Applikationen, wird in ebenso tiefgehender Ausführlichkeit im zweiten Teil des Buches präsentiert.

Verschiedene Gruppen von Lesern kommen bei diesem Buch auf Ihre Kosten: sowohl fortgeschrittene Anfänger im Bereich der Programmierung als auch erfahrene Web-Programmierer, die ihr Wissen updaten wollen. Das Buch „JavaScript. The Definitive Guide“ ist ein sehr ausführliches und kompetentes Werk, dessen Anspruch auf Aktualität vom Autor sehr ernst genommen wird (vgl. z.B. die explizite Orientierung an die Anforderungen von HTML5 oder die Vorstellung bestimmter Applikationen, die noch so frisch sind, dass sie zum Zeitpunkt des Schreibens des Buches noch nicht von allen Browsern unterstützt wurden, aber als zukuntsorientiert anzusehen sind.)

Am wichtigesten ist jedoch, dass man auf jeden Fall bei diesem Buch einen tiefen Einblick in das große, faszinierende und umfangreiche JavaScript-Reich gewinnt, einschließlich der Welt der Frameworks und Hilfstools, die uns helfen, JavaScript leichter und effektiver einzusetzen.

Eben ein wirklicher *Definitive Guide*, der diesen Namen verdient hat.