04 – Die Sprache JavaScript im Detail

I3B

Leon Raab

1822936

Inhalt

1 Was ist JavaScript?

Dieses Kapitel beschäftigt sich mit dem Grundbegriff JavaScript und dessen Einordnung, sowie einem kurzen historischen Abriss über die Entwicklung der Sprache. Abschließend werden grundlegende Merkmale und Eigenschaften vorgestellt.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *das Anwendungsgebiet von JavaScript darzustellen und einzuordnen,*
- *zu wissen, wie sich JavaScript entwickelt hat und was diese Entwicklung maßgeblich beeinflusste, sowie*
- *grundlegende Merkmale der Sprache wie den Aufbau und die Struktur grob zu kennen.*

1.1 Begriff

JavaScript ist heutzutage eine der meist verwendeten Programmiersprachen. Als Einsatzgebiet gilt in erster Linie die webbezogene Programmierung. Hierbei ist zu beachten, dass JavaScript eine Art Ergänzung zu anderen Sprachen darstellt. Sie tritt also nur sehr unwahrscheinlich alleinstehend auf, sondern bildet einen der Bausteine zur Entwicklung einer Webseite oder Anwendung.

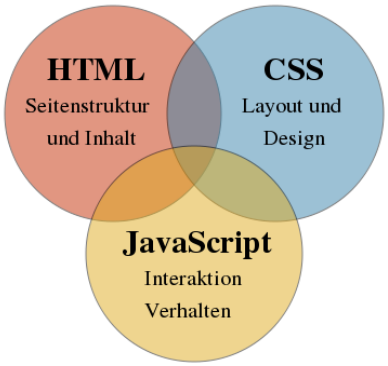

Abb. 1.1: Die Bausteine HTML, CSS und JavaScript (Quelle: https://wiki.selfhtml.org/images/thumb/7/78/HTML-CSS-JS.svg/400px-HTML-CSS-JS.svg.png)

In der Abbildung ist zu erkennen, wie das Zusammenspiel verschiedener Technologien aufgebaut ist, um letztendlich eine Webseite zu erzeugen. So bestimmt HTML die Struktur und den Inhalt der Seite, CSS ist für das Layout und die Designmerkmale zuständig. Die Hauptaufgabe von JavaScript besteht darin, den HTML Code einer Seite zu unterstützen und dynamischer zu machen. Dies erfolgt durch diverse Funktionen wie den Inhalt einer Seite clientseitig zu generieren, ohne dass die Seite jedes Mal neu angefordert und geladen werden muss. Durch Auswerten des Verhaltens des Benutzers oder die clientseitige Überprüfung von Formularen wird somit eine Webseite mit der nötigen Interaktivität versorgt, welche heutzutage von den meisten Benutzern gewünscht oder sogar vorausgesetzt ist.

JavaScript ist zudem eine interpretierende Sprache, wobei der Internet-Browser den Interpreter darstellt. Hierbei sind alle gängigen Browser dazu in der Lage, JavaScript zu interpretieren.

1.2 Historischer Abriss

Nachfolgend gibt es zunächst einen kurzen historischen Abriss zur Entwicklung des heutigen JavaScripts. Die Sprache ist circa zwischen 1995 und 1996 von Brendan Eich von Netscape Communications entwickelt worden, wobei sie zunächst den Namen *Mocha* und später *LiveScript* trug. Die Marketingabteilung der Firma entschloss sich kurze Zeit später, es wäre eine gute Idee, die derzeitige Popularität der Sprache Java zu nutzen und daraus Profit zu schlagen, weswegen sie sich

dazu entschieden, die Sprache letztendlich in JavaScript umzubenennen. Darauf folgte jedoch Verwirrung bei den Anwendern, da JavaScript im Grunde nur sehr wenig mit Java zu tun hatte und auch die Anwendungsgebiete unterschiedlich waren [vgl. SS11].

Trotz all dem erkannten Programmierer bereits früh die oben erwähnten Vorteile und die Nützlichkeit der Sprache gegenüber reinem HTML. Unterstützt wurde die zunehmende Popularität vor allem durch die Einbindung in den zu der Zeit weit verbreiteten Netscape Navigator (seit Version 2.0 [SK09]). Zwar war die erste Implementierung im Browser noch wenig zufriedenstellend, es wurde jedoch stetig verbessert und weiterentwickelt.

Maßgeblich beeinflusst wurde die Entwicklung von JavaScript zudem durch einen starken Konkurrenzkampf zwischen JavaScript von Netscape und der parallel entwickelten Sprache JScript von Microsoft und deren Internet Explorers. Jahrelang gab es ein sprichwörtliches Wettrennen der beiden Anbieter. Hierbei entstand zudem mit der European Computer Manufacturers Association ab 1997 das sogenannte ECMAScript.

Definition 1.1: ECMAScript

Das ECMAScript legt fest, wie ein JavaScript System zu funktionieren hat, um die Unterschiede möglichst gering zu halten. Somit bildet es den Sprachkern von JavaScript und wird auch oft als offizieller Name verwendet.

Es legt beispielsweise die verschiedenen Sprachelemente wie Datentypen oder Kontrollstrukturen und Sicherheitsaspekte fest. Der Standard sagt jedoch nichts über die letztendliche Einbindung in einen Webbrowser aus. Hierfür gibt es andere Standards, wie zum Beispiel das Document Object Model (DOM), auf das später noch genauer eingegangen wird. Auch diese Dokumente wurden und werden mit der Zeit stetig weiterentwickelt und alle populären Browser haben sie implementiert. Die aktuelle Version hat die Bezeichnung *ECMAScript 2019*.

1.3 Merkmale und Eigenschaften

Definition 1.2: JavaScript-Programm

Ein JavaScript-Programm besteht grundsätzlich aus einer oder mehreren Anweisungen. Diese werden wiederum aus verschiedenen Teilen zusammengesetzt, und zwar Schlüsselwörter, Operatoren, Bezeichner und Literale.

Ein kleines Beispiel, um diese Teile zu erklären sieht wie folgt aus:

```
1   var simpleNumber = 6;
```

Codebsp. 1.1: Einfache JavaScript-Zuweisung

Diese Anweisung beginnt mit einem Schlüsselwort (*var*), gefolgt von einem Bezeichner (*simpleNumber*), einem Operator (=) und letztendlich einem Wert (*6*). Weiterführend werden Anweisungen dann in einer bestimmten Anordnung im Programm festgelegt, wodurch eine oder mehrere Funktionen ausgeführt werden können. Zudem gibt es von JavaScript fest definierte Funktionen, die in Programmen verwendet werden können. Einige von diesen werden im Folgenden noch vorgestellt.

In JavaScript kann man sowohl prozedural entwickeln, also über einfache Funktionen wie oben beschrieben, es besteht aber auch die Möglichkeit, einen objektorientierten Programmieransatz zu wählen, was jedoch ein wenig komplexer ist.

Eine weitere Eigenschaft der Sprache ist die Typisierung. Diese erfolgt vollständig dynamisch, die Variablen haben also bei der Deklaration keinen bestimmten Datentyp, sondern erst zur Laufzeit, sobald sie mit echten Werten instanziiert und belegt sind.

Hauptanwendungsgebiet von JavaScript ist die Client-Seite (CSJS), das bedeutet, dass JavaScript auf der Seite des Benutzers, also auf dem Computer oder mobilen Endgerät, im Browser ausgeführt wird. Daneben gibt es auch serverseitiges JavaScript (SSJS), dieses unterscheidet sich durch den anderen Einsatzzweck nur geringfügig vom clientseitigen JavaScript, wird jedoch im Gegensatz zu anderen Sprachen wie Perl oder PHP nur selten verwendet.

Zusammenfassung

In diesem Kapitel wurde zunächst der Begriff JavaScript erklärt und die Bedeutung der Sprache beschrieben.

Anschließend wurde in einem kurzen historischen Abriss die Entwicklung dargestellt und wichtige Ereignisse, wie das Erstellen verschiedener Standards, aufgezeigt.

Abschließend wurden die wichtigsten Merkmale und Eigenschaften kurz beschrieben und erklärt, wobei im Laufe des Skriptes an den jeweiligen Stellen weiterführend auf die Technologien und Funktionsweisen eingegangen wird.

Aufgaben zur Selbstüberprüfung

Aufgabe 1.1:

Was sind die Einsatzgebiete von JavaScript und wofür wird es hauptsächlich verwendet?

Aufgabe 1.2:

Was ist das ECMAScript und was legt es fest?

Aufgabe 1.3:

Nennen sie eine beispielhafte JavaScript Anweisung zur Zuweisung einer Variablen.

2 Integration in HTML

Dieses Kapitel beschäftigt sich mit der Integration von JavaScript in HTML, wobei auf die verschiedenen Möglichkeiten hierzu eingegangen wird. Zur Veranschaulichung werden diese kurz in Codebeispielen gezeigt und erklärt.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *die verschiedenen Arten der Integration von JavaScript in HTML zu nennen,*
- *zu wissen, wie man diese im Code einbindet und*
- *einige Vorteile und Nachteile der jeweiligen Möglichkeiten nennen und begründen können.*

2.1 Möglichkeiten der Einbindung in HTML

Da JavaScript in HTML integriert wird, arbeitet man hauptsächlich mit HTML-Dateien. JavaScript-Anweisungen können hierbei an mehreren Stellen untergebracht und eingebunden werden. Eine Möglichkeit ist das direkte Codieren in den HTML-Code zwischen den Tags <*script*> und </*script*>. Alternativ kann der JavaScript-Code auch in einer externen Datei gespeichert werden, welche später im HTML-Dokument eingebunden wird. Weiterhin können in den HTML-Code JavaScript-Links eingebaut werden, welche mit Code hinterlegt sind. Es gibt zudem Attribute, welche als Event-Handler fungieren können, die also auf bestimmte Ereignisse durch den Benutzer reagieren, indem sie mit Code hinterlegt sind.

2.1.1 Verwendung von <*script*>

Die naheliegendste Methode, JavaScript-Anweisungen auszuführen, ist die Verwendung des <*script*> Tags. Hierbei ist es grundsätzlich unwichtig, wo genau dies im HTML-Code passiert. Zu beachten ist jedoch, dass die Einbindung an vielen verschiedenen Stellen den Code insgesamt unübersichtlich machen kann, falls das der Fall ist, sollte man ihn möglichst in den *head* schreiben. Hier ist es nicht nötig. Es folgt ein erstes Beispiel zur Verwendung des Tags innerhalb eines HTML-Dokuments:

```
1  <html>
2      <head>
3          <title>Erstes Beispiel</title>
4      </head>
5      <body>
6          <script>
7              alert("Dies ist das erste Beispiel!")
8          </script>
9      </body>
10 </html>
```

Codebsp. 2.1: JavaScript im <*script*> Tag

Hier wurde der beispielhafte JavaScript-Code direkt in den HTML-body eingebaut, es wird im Browser durch den Befehl *alert(message)* eine Meldung mit dem Text „Dies ist ein erstes Beispiel“ angezeigt.

Zu beachten ist hierbei, dass der JavaScript-Code direkt ausgeführt wird, sobald beim Laden des HTML-Dokuments das <*script*> Tag erkannt wird, was bei großen Programmen zu Problemen und langen Ladezeiten führen kann.

2.1.2 Einbinden einer externen Datei

JavaScript-Code kann auch in einer JavaScript-Bibliotheksdatei programmiert werden, womit der komplette Code in eine separate Datei ausgelagert wird. Diese Datei muss dann immer die Endung *.js* besitzen. Somit muss in der HTML-Datei im <*script*> Tag nur noch der Ort angegeben werden, unter dem diese zu finden ist. Dies wird im folgenden Codebeispiel gezeigt, wobei zur Einfachheit der restliche HTML-Code weggelassen wird. Dieser ist der gleiche Code wie in Codebeispiel 2.1.

```
1  // erste externe JavaScript-Datei
2  alert("Dies ist das zweite Beispiel!");
```

Codebsp. 2.2: Erste externe JavaScript-Datei „beispiel2.js“

```
6          <script src="beispiel2.js"></script>
```

Codebsp. 2.3: Einbinden der externen JavaScript-Datei

Wie im Beispiel gezeigt ist, wird die erstellte JavaScript-Datei innerhalb des <*script*> Tags geladen, wobei im Attribut *src* der Name der Datei angegeben wird.

Der größte Vorteil des Einbindens einer externen Datei besteht darin, dass man JavaScript-Programme, die auf mehreren unterschiedlichen Seiten und an verschiedenen Stellen benötigt werden, nicht mehrmals ausprogrammieren muss. Man kann sie an den jeweiligen Stellen einfach einbinden, somit beschränkt sich auch der Aufwand bei Änderungen im Code auf eine einzige Änderung in der JavaScript-Datei. [vgl. CW10]

2.1.3 JavaScript-Links

Bei den ersten beiden vorgestellten Methoden (2.1.1 und 2.1.2) wird der JavaScript-Code beim Laden des HTML-Dokuments direkt ausgeführt. Oftmals werden JavaScript-Befehle jedoch aufgrund von Benutzereingaben ausgeführt. So auch im folgenden Beispiel, bei dem der Code in einen HTML-Link verpackt wird. Er wird ausgeführt sobald der Benutzer auf den Link klickt.

```
6          <a href="javascript:alert('Dies ist das dritte Beispiel!');">Beispiel 3<a>
```

Codebsp. 2.4: Verwendung eines JavaScript-Links

Wie im Beispiel zu sehen ist, verweist die URL im HTML-Element *Anchor* auf einen Link mit dem Protokoll JavaScript (*javascript:*). Klickt der Benutzer auf den Link mit dem Namen „Beispiel 3“, so wird ihm also, sofern sein Browser JavaScript unterstützt, eine Meldung angezeigt mit der Nachricht „Dies ist das dritte Beispiel!“.

2.1.4 JavaScript in Event-Handlern

Auch bei dieser Art der Einbindung wird der JavaScript-Code nicht einfach direkt beim Laden ausgeführt, sondern das Ziel ist es, auf das Verhalten und die Eingaben des Benutzers einzugehen. Es gibt verschiedene HTML-Elemente, welche JavaScript-Code enthalten können. Dieser wird dann bei bestimmten Ereignissen (engl.: „events“) ausgeführt. Im folgenden Beispiel wird dies anhand eines einfachen Buttons gezeigt.

```
6              <button onclick="alert('Dies ist das vierte Beispiel!');">Beispiel 4</button>
```

Codebsp. 2.5: Button mit JavaScript Event-Handler

Es wird ein Button erstellt und der Event-Handler definiert, indem festgelegt wird, was bei Benutzerinteraktion passieren soll. In diesem Fall wird eine Meldung angezeigt, wenn der Benutzer auf den Button klickt. Der Code steht also hinter dem Attribut *onclick*. Neben diesem vorgestellten Event-Handler, um auf ein Anklicken zu reagieren, gibt viele weitere mit unterschiedlichen Auslösern. Grundsätzlich beginnen diese stets mit *on*.

Zusammenfassung

Dieses Kapitel hatte zum Ziel, die verschiedenen Arten, JavaScript in HTML einzubinden, aufzuzeigen.

Weiterhin wurden diese Möglichkeiten einzeln erläutert und an Codebeispielen dargelegt. Hierbei wurden zudem mögliche Vor- und Nachteile beschrieben.

Aufgaben zur Selbstüberprüfung

Aufgabe 2.1:

Was sind die vier verschiedenen Möglichkeiten, JavaScript in HTML einzubinden?

Aufgabe 2.2:

Mit welchem Tag kann JavaScript in HTML eingebunden werden?

Aufgabe 2.3:

Mit welchem Attribut kann ein externes JavaScript-Programm in HTML eingebunden werden?

Aufgabe 2.4:

Welches Protokoll wird benötigt, um JavaScript in HTML-Links einzubauen und wie ist ein solcher Link aufgebaut?

Aufgabe 2.5:

Wie kann man mit JavaScript auf Benutzereingaben oder dessen Verhalten eingehen?

3 Variablen, Datentypen und Merkmale

Dieses Kapitel beschäftigt sich zunächst mit Variablen in JavaScript und die Möglichkeiten, diese zu deklarieren. Weiterhin werden die verschiedenen Datentypen vorgestellt und behandelt. Folgend wird die Funktionsweise des Scopes in JavaScript erläutert. Zuletzt wird der Begriff Hoisting und dessen Nutzen in JavaScript dargelegt.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *die verschiedenen Möglichkeiten, Variablen in JavaScript zu deklarieren, anwenden können,*
- *zu wissen, welche Datentypen es gibt und wie man diese erzeugt,*
- *zu wissen welche Arten von Scopes es in JavaScript gibt, von wo man auf welchen Scope zugreifen kann und wann eine Variable welchen Scope besitzt,*
- *den Begriff Hoisting erklären und verwenden zu können,*
- *zu wissen, was die Truthiness aussagt und*
- *das Konzept und die Funktionsweise von Typumwandlungen in JavaScript erklären zu können.*

3.1 Variablen

Variablen können in JavaScript auf drei unterschiedliche Arten deklariert werden. Die erste und verbreitetste Variante ist das Schlüsselwort *var*. Die zweite Möglichkeit ist die Verwendung von *let*, wobei dies in der Praxis eher seltener verwendet wird. Die letzte Möglichkeit unterscheidet sich etwas zu den anderen, denn unter Verwendung von *const* kann man eine Art Konstante deklarieren, also eine Variable, die unveränderlich ist.

Eine Variable kann entweder ohne Wert instanziiert werden oder gleich mit einem Anfangswert versehen werden (vgl. Codebsp. 1.1). Nachdem eine Variable erstellt wurde, kann mit dessen Namen weitergearbeitet werden. Bei der Namensgebung sind einige Regeln und Konventionen zu beachten. Diese werden im Folgenden ausgeführt.

Alle Variablen müssen mit einem Buchstaben beginnen und dürfen keine Leerzeichen enthalten. Sie dürfen insgesamt nur aus Buchstaben und Ziffern bestehen, wobei das Sonderzeichen „_“ erlaubt ist, keine Umlaute oder *ß* enthalten und nicht mit einem Schlüsselwort identisch sein.

Alles weitere sind Namenskonventionen, welche sich von Entwickler zu Entwickler unterscheiden können. So ist es beispielsweise weit verbreitet, die sogenannte *camelCase*-Notation zu verwenden.

3.2 Datentypen

JavaScript arbeitet mit einer schwachen Typisierung, das bedeutet, dass in einer Variablen alle Arten von Werten gespeichert werden können, also auch verschiedene Typen hintereinander. Falls dies der Fall ist, werden eventuell notwendige Typänderungen im Hintergrund automatisch vorgenommen. Der Datentyp richtet sich also zur Laufzeit immer nach dem tatsächlichen Wert, der in ihr gespeichert ist. Somit haben nicht initialisierte Variablen den Typ *undefined*.

Bei den Datentypen wird unterschieden zwischen primitiven Datentypen, diese sind *Boolean*, *String* und *Number*, und Objekt-Typen, zu diesen zählen *Function*, *Array*, *Date*, *Object* und *RegExp*. Auf diese speziellen Typen wird im Späteren noch genauer eingegangen.

Zur Erzeugung der Datentypen stehen zwei verschiedene Möglichkeiten zur Auswahl, wobei diese nicht für alle gleich gelten. Primitive Datentypen sowie Objekte können sowohl über Literalte, also

durch einfache Zuweisung durch den „=“-Operator, als auch über einen Konstruktor erstellt werden. Für alle anderen Datentypen stehen nur Konstruktoren zur Verfügung. Es folgen einige Beispiele zur Veranschaulichung dieser Möglichkeiten.

```
1  var numberLiteral = 4;
2  var stringLiteral = "BeispielString"
3  var booleanLiteral = true;
4  var objectLiteral = {numberLiteral: 19, stringLiteral:"Leon"};
```

Codebsp. 3.1: Variablenerzeugung mit Literalen

In Codebeispiel 3.1 wird gezeigt, wie nacheinander jeweils eine Variable des Typs *Number*, *String*, *Boolean* und einem *Object* durch Zuweisung an ein Literal erzeugt werden.

```
1  var stringInstance = new String("4");
2  var numberInstance = Number(stringInstance);
3  var booleanInstance = Boolean(true);
4  var objectInstance = new Object();
5  var arrayInstance = new Array(1, 2, 3);
6  var dateInstance = new Date("November 1, 2019 12:00:00");
```

Codebsp. 3.2: Variablenerzeugung mit Instanzen

In diesem Codebeispiel ist zu sehen, wie verschiedene Datentypen über Konstruktoraufrufe neu erzeugten Instanzen zugeordnet werden. Jedoch ist zu beachten, dass, vor allem für die primitiven Datentypen, die Erzeugung über Literale zu bevorzugen ist.

3.3 Scope

Es gibt in JavaScript zwei unterschiedliche Arten von Scopes, den lokalen Scope und den globalen Scope. Dieser legt die Möglichkeiten, auf eine Variable zuzugreifen fest, also deren Sichtbarkeit.

In JavaScript erzeugt jede Funktion einen neuen Scope. Das heißt, auf Variablen, die innerhalb der Funktion neu erstellt werden, kann von außen nicht zugegriffen werden. Innerhalb der Funktion kann man jedoch auf alle Variablen, die außerhalb dieser erzeugt wurden, zugreifen. Alle nicht in einer Funktion definierten Variablen liegen somit im globalen Scope. Weiterhin werden alle lokalen Variablen gelöscht, sobald die Funktion, in der sie liegen, abgeschlossen ist oder beendet wird. Globale Variablen hingegen werden erst gelöscht, sobald das Programm beendet ist, also im Anwendungsfall meist, wenn der Webbrowser geschlossen wird.

```
1  var globalScope = 5;
2  function scopeFuntion(){
3      var localScope = 4;
4  }
```

Codebsp. 3.3: Variablen mit globalen und lokalen Scopes

Im Beispiel werden zwei verschiedene Variablen deklariert, eine globale (Zeile 1) und eine lokale (Zeile 3), welche nur in der Funktion *scopeFunction* sichtbar ist. Auf die globale Variable kann also auch innerhalb der Funktion zugegriffen werden.

3.4 Hoisting

Definition 3.1: Hoisting

Mit Hoisting (deutsch: „hochziehen“) wird in JavaScript das Verhalten des Interpreters bezeichnet, bei dem alle Variablendeklarationen an den Anfang des derzeitigen Scopes gezogen werden. Somit kann eine Variable schon verwendet werden, bevor sie überhaupt deklariert wurde.

```
function hoistingExample(){
    x = 5;
    alert(x);
    var x;
}
```

Codebsp. 3.4: Beispiel zum Hoisting

Wie im Beispiel zu erkennen ist, wird x am Anfang des Codes belegt und danach in einer Meldung verwendet. Deklariert wird es, zumindest im Code, jedoch erst nachdem es verwendet wird. Dies ist dadurch möglich, weil der Interpreter die Deklaration bei der Ausführung intern an den Anfang setzt.

Es gibt jedoch ein paar wichtige Dinge zu beachten. Zum einen funktioniert das Hoisting nur bei Variablen, die mit *var* deklariert werden, mit *let* und *const* funktioniert es nicht. Zum anderen werden in JavaScript nur Deklarationen gehoisted, nicht aber Initialisierungen.

Trotz des Hoistings ist es am übersichtlichsten und für andere am leichtesten verständlich, Variablen am Beginn jedes Scopes zu deklarieren.

3.5 Truthiness

In JavaScript gibt es wie bereits erläutert den Datentyp *Boolean* für Wahrheitswerte, er kann also entweder *true* oder *false* sein. Jeder Ausdruck kann von JavaScript ausgewertet werden und das Ergebnis davon, der Wahrheitswert, wird durch die Truthiness festgelegt.

Im Grunde lässt sich sagen, dass alles, was einen echten Wert besitzt, zu *true* ausgewertet wird. Beispiele hierfür sind unter anderem Zahlen, Strings oder Objekte.

Hingegen wird alles was keinen Wert besitzt zu *false* ausgewertet. Dazu zählen die Zahlen 0 und -0, der Leerstring ““, der Wahrheitswert *false* und die besonderen Werte *undefined*, *null* und *NaN* (Not a Number).

3.6 Typumwandlung

Typumwandlungen können auf zwei verschiedene Arten ablaufen. In den meisten Fällen werden Werte falls benötigt von JavaScript automatisch implizit umgewandelt. Dies passiert immer dann, wenn Operatoren oder Funktionen mit Werten aufgerufen werden, die nicht den erwarteten, also vorgegebenen, Datentypen entspricht. Als Beispiel hierfür kann wieder die Funktion *alert* herangezogen werden, denn alle Werte, die man der Funktion übergibt, werden implizit zu einem String umgewandelt. Man kann die Funktion also zum Beispiel mit einer beliebigen Zahl aufrufen und bei Ausführung des Codes wird diese zu einem String konvertiert. Hierbei ist zu beachten, dass die Typumwandlung, wenn möglich, immer zum größeren Datentyp hin passiert, um zum Beispiel bei Zahlen keine Genauigkeitsverluste verzeichnen zu müssen.

Die zweite Möglichkeit ist die explizite Typumwandlung des Programmierers selbst. Alle Werte lassen sich manuell zu einer Zahl mit dem Number-Konstruktor, zu einem String mit dem String-Konstruktor oder der Methode *toString()* oder zu einem Boolean mit dem Boolean-Konstruktor umwandeln, wie das folgende Beispiel zeigt.

```
var numberToString = String(5);
var dateToString = Date().toString();
var booleanToNumber = Number(true);
var numberToBoolean = Boolean(0);
```

Codebsp. 3.5: Explizite Typumwandlungen

Zusammenfassung

Dieses Kapitel hatte zum Ziel, den Umgang mit der Deklaration von Variablen aufzuzeigen. Weiterhin sollte ein Verständnis für die verschiedenen Datentypen in JavaScript aufgebaut werden. Hierzu wurden Unterschiede in der Erzeugung erläutert.

Des Weiteren wurde der Wertebereich (Scope) von Variablen unterlegt mit einem Codebeispiel dargestellt und der Begriff des Hoistings definiert und dessen Funktionsweise erläutert. Folgend wurde das Konzept der Truthiness in JavaScript erläutert und beschrieben, welche Werte wie ausgewertet werden. Zuletzt beschäftigte sich dieses Kapitel mit der Art und Weise der Typumwandlungen in JavaScript und wieso man diese benötigt.

Aufgabe 3.1:

Welche Möglichkeiten gibt es, in JavaScript eine Variable zu deklarieren?

Aufgabe 3.2:

Welchen Datentyp hat eine Variable und wann?

Aufgabe 3.3:

Welche Sichtbarkeitsbereiche gibt es in JavaScript und wann hat eine Variable welchen Sichtbarkeitsbereich?

Aufgabe 3.4:

Was versteht man in JavaScript unter dem Begriff „Hoisting“?

Aufgabe 3.5:

Was beschreibt die Truthiness in JavaScript?

Aufgabe 3.6:

Wie läuft in JavaScript die Typumwandlung ab?

4 Kontrollstrukturen

Dieses Kapitel beschäftigt sich mit den verschiedenen Kontrollstrukturen in JavaScript. Zunächst werden die Verzweigungen if und switch erklärt und danach wird auf verschiedene Schleifen eingegangen. Zu diesen gehören while, for, for…in und for…of. Die Kontrollstrukturen werden weiterhin an Beispielen aufgezeigt.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *verschiedene Arten von Verzweigungen zu kennen und diese anwenden zu können und*
- *die verschiedenen Schleifen und deren jeweilige Funktionsweise zu kennen.*

4.1 Verzweigungen

4.1.1 *if*-Anweisung

Wenn eine Aktion beziehungsweise ein Codeblock nur unter einer bestimmten Bedingung ausgeführt werden, wird hierzu der *if*-Operator verwendet. Er prüft, ob der Ausdruck mit *true* ausgewertet wird. Im folgenden Beispiel wird die Meldung innerhalb des *if*-Blocks ausgegeben.

```
1  var x = 5;
2  if(x === 5){
3      alert("Richtig!");
4  }
```

Codebsp. 4.1: if-Verzweigung

Zu beachten ist hierbei, dass es einen Unterschied zwischen den Vergleichsoperatoren == und === gibt. Das doppelte Gleichzeichen konvertiert die Werte zuerst zu einem gemeinsamen Typ (vergleiche Kapitel 3.6), das dreifache Gleichzeichen hingegen vergleicht die Werte direkt.

Es gibt weiterhin Erweiterungen zum einfachen *if*. Mit *else if* kann eine weitere Bedingung angegeben werden, die ausgeführt wird, falls die erste Bedingung falsch ist. Mit *else* kann ein Codeblock ausgeführt werden, falls alle vorherigen if-Bedingungen falsch waren.

4.1.2 *switch*-Anweisung

Mit einer switch-Anweisung kann einer von vielen Code-Blöcken zur Ausführung ausgewählt werden. Hierbei wird die switch-Bedingung einmal ausgewertet und das Ergebnis wird mit den Werten jedes Falls verglichen. Falls die Werte übereinstimmen, wird der jeweilige Code ausgeführt.

```
1  var produkt = "Apfel";
2  switch(produkt){
3      case "Banane":
4          alert("1€");
5          break;
6      case "Birne":
7          alert("1.5€");
8          break;
9      case "Apfel":
10         alert("0.5€");
11         break;
12     default:
13         alert("Nicht verfügbar.");
14 }
```

Codebsp. 4.2: *case*-Anweisung

Es wird eine Meldung mit „0.5€“ ausgegeben. Durch das *break* Statement wird der switch-Block direkt nach Fund eines passenden Wertes verlassen. Falls kein passender Wert gefunden wird, wird das *default* Statement ausgeführt.

4.2 Schleifen

4.2.1 while und do…while-Schleife

Eine *while*-Schleife führt den darin enthaltenen Code solange aus, wie die darin enthaltene Bedingung wahr ist. Zu beachten sind hierbei zwei Aspekte. Zum einen kann es durchaus sein, dass der Code in einer while-Schleife nie ausgeführt wird. Im Gegensatz dazu gibt es die *do...while*-Schleife, welche den eingeschlossenen Code mindestens einmal ausführt, erst dann wird die Bedingung zum ersten Mal geprüft. Zum anderen sollte immer daran gedacht werden, die zu testende Bedingung innerhalb der Schleife zu verändern. Ansonsten erhält man eine Endlosschleife, da die Bedingung immer wahr ist.

```
1  var count = 0;
2  do {
3     alert("Count ist: " + count);
4     count++;
5  } while (count < 0);
```

Codebsp. 4.3: *do…while*-Schleife

Im obigen Beispiel wird die Meldung einmal ausgegeben, obwohl die Bedingung von Anfang an nicht erfüllt ist. Wäre derselbe Code in einer einfachen *while*-Schleife, würde er nie ausgeführt werden.

4.2.2 *for*-Schleife

Eine *for*-Schleife wird häufig in der gleichen Art genutzt, wie eine *while*-Schleife. Der Unterschied besteht darin, dass die Bedingung in kompakter Form initialisiert, überprüft und verändert werden kann. Die Anweisung enthält hierzu drei Klauseln. In der ersten Klausel wird der Anfangsausdruck festgelegt, also meist eine Variable initialisiert. In der zweiten Klausel wird eine Bedingung mit der besagten Variablen überprüft und in der letzten Klausel wird sie für die nächste Überprüfung verändert.

```
1  for (var count = 0; count < 1; count++){
2      alert("Count ist: " + count);
3  }
```

Codebsp. 4.4: *for*-Schleife

Das Beispiel erzeugt also dieselbe Ausgabe wie im Codebeispiel 4.3.

4.2.3 for...in und for…of-Schleife

Mit der *for…in*-Schleife werden alle Eigenschaften eines Objekts in willkürlicher Reihenfolge durchlaufen. Der Code innerhalb der Schleife wird dann für jede Eigenschaft einmal ausgeführt. Das Grundprinzip ist für die *for…of*-Schleife dasselbe, wobei diese über die Eigenschaften oder Werte eines aufzählbaren Objektes, wie zum Beispiel einem Array, iteriert. Dabei werden die Eigenschaften in der richtigen Reihenfolge, also am Beispiel des Arrays anhand des Indexes, durchlaufen.

```
1  var string = "Einzeln"
2  var c;
3  for(c of string){
4      alert(c);
5  }
```

Codebsp. 4.5: *for…of*-Schleife

Im Beispiel wird der String über die Stelle der Zeichen iteriert, also wird als Ergebnis das Wort „Einzeln“ in einzelnen Buchstaben ausgegeben.

Zusammenfassung

Dieses Kapitel hatte zum Ziel, die verschiedenen Arten der Kotrollstrukturen zu erläutern.

Hierbei wurde zunächst auf die verschiedenen Verzweigungen eingegangen, danach auf die verschiedenen Schleifenarten. Beides wurde mit mehreren Beispielen unterlegt.

Aufgabe 4.1:

Welche Arten von Verzweigungen gibt es in JavaScript?

Aufgabe 4.2:

Welche Arten von Schleifen gibt es in JavaScript?

5 Arrays

Dieses Kapitel beschäftigt sich mit Arrays in JavaScript. Zunächst werden deren Eigenschaften und Funktionsweise erläutert, danach folgen einige nützliche Methoden, um mit Arrays zu arbeiten.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *ein Array in JavaScript zu erstellen und mit Werten zu füllen und*
- *die wichtigsten Funktionen auf Arrays zu kennen und sie einsetzen zu können.*

5.1 Eigenschaften

Arrays werden in JavaScript verwendet, um mehrere Werte in einer einzigen Variablen zu speichern. Auf diese Werte kann dann über den Variablennamen und dem gewünschten Index zugegriffen werden. Wie in den meisten anderen Programmiersprachen beginnt der Index bei 0.

Standardmäßig wird ein Array in JavaScript direkt bei der Deklaration durch eckige Klammern mit Werten gefüllt, wie im folgenden Beispiel gezeigt ist. Da die Länge eines JavaScript-Arrays veränderbar ist, können später beliebig Werte hinzugefügt oder entfernt werden.

```
1  var testArray = ["Beispiel", "Array"];
2  testArray[2] = "!";
```

Codebsp. 5.1: Ein Array in JavaScript

Alternativ kann ein Array auch über den Konstruktor *new Array()* deklariert und später mit Werten gefüllt werden.

In JavaScript können Arrays sowohl primitive Datentypen als auch Objekte speichern. Weiterhin können in einem Array verschiedene Datentypen gespeichert werden.

5.2 Methoden

JavaScript bietet viele verschiedene nützliche Methoden, um mit Arrays zu arbeiten. Diese müssen am jeweiligen Array aufgerufen werden.

Mit der Methode *length* wird die Anzahl der Elemente im Array zurückgeliefert.

Mit *sort* werden die Elemente im Array, sofern möglich, sortiert.

Mit der Funktion *forEach(…)* wird jedes Element des Arrays durchlaufen und an diesem der Ausdruck in der Klammer ausgeführt.

Mit *push(…)* wird das Element in den Klammern an letzter Stelle des Arrays hinzugefügt.

Zusammenfassung

Dieses Kapitel sollte einen kleinen Überblick über Arrays in JavaScript geben.

Zunächst wurde deren Funktionsweise und Eigenschaften aufgezeigt und danach einige nützliche Methoden vorgestellt, um mit ihnen zu arbeiten.

Aufgabe 5.1:

Wie wird in JavaScript standardmäßig ein Array erstellt? Erstellen sie ein Array „autos“ mit zwei Beispielautos als Strings.

Aufgabe 5.2:

Wie kann man die Anzahl der gespeicherten Werte in einem Array ausgeben lassen?

6 Funktionen

Dieses Kapitel beschäftigt sich mit Funktionen in JavaScript. Zunächst wird der Begriff definiert, danach im Kontext von JavaScript beleuchtet. Die Syntax wird anhand eines Codebeispiels gezeigt und wichtige Merkmale dargelegt.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *das Konzept und die Grundidee von Funktionen darlegen zu können,*
- *den Aufbau von Funktionen in JavaScript zu kennen und*
- *Funktionen in JavaScript selbst definieren und aufrufen zu können*

6.1 Begriff, Syntax und Eigenschaften

Definition 6.1: Funktion

Mit einer Funktion wird eine Reihe von Anweisungen bezeichnet, die zum Ziel hat, eine gewisse Aufgabe auszuführen oder einen Wert zu berechnen. Dabei kann sie im Code an allen Stellen im gleichen Scope aufgerufen werden.

[vgl.: https://developer.mozilla.org/de/docs/Web/JavaScript/Guide/Funktionen]

In JavaScript gibt es verschiedene Arten von Funktionen. Zunächst gibt es die objektunabhängigen Funktionen, diese sind in JavaScript bereits vordefiniert. Ein Beispiel hierfür ist wieder *alert(…)*. Des Weiteren gibt es selbst definierte Funktionen, wie das folgende Codebeispiel zeigt.

Eine Funktion wird in JavaScript mit dem Schlüsselwort *function* definiert, darauf folgen der Name der Funktion und Klammern, welche optional formale Parameter enthalten können. In den geschweiften Klammern wird dann der Code geschrieben, der beim Aufruf ausgeführt werden soll.

```
1 function beispielFunktion(eingabe){
2     alert(eingabe);
3 }
4 var beispielText = "Erste Funktion!"
5 beispielFunktion(beispielText);
```

Codebsp. 6.1: Beispielfunktion

Im Beispiel wird eine Funktion definiert, die einen Eingabeparameter hat. Diese wird dann ausgegeben, wenn sie aufgerufen wird. Funktionsnamen sollten immer mit Kleinbuchstaben beginnen und können Buchstaben, Zahlen oder Unterstriche enthalten. Die Funktion wird dann mit dem String *beispielText* aufgerufen, dieser wird dann in einer Meldung angezeigt. Nun ist klar, dass *alert(…)* auch eine ganz normale Funktion mit einem Eingabeparameter ist.

Der Vorteil von Funktionen besteht darin, Code, der einmal definiert wurde, beliebig oft ausführen zu können. Zur Laufzeit ist eine Funktion nichts anderes als ein Objekt. Deshalb kann zum Beispiel auch eine Funktion referenziert werden, indem die Klammern hinter dem Aufruf weggelassen werden.

Es gibt verschiedene Szenarien, durch die eine Funktion ausgeführt wird. Sie kann durch anderen JavaScript Code aufgerufen werden, durch ein Event ausgelöst werden (mehr dazu später) oder von sich selbst aufgerufen werden. Mit dem Schlüsselwort *return* wird die Funktion beendet und es kann ein Wert an den Aufrufer zurückgegeben werden.

In JavaScript ist es durchaus möglich, Funktionen innerhalb von anderen Funktionen zu deklarieren, diese sind dann natürlich im lokalen Scope der äußeren Funktion. Weiterhin gibt es die Möglichkeit, anonyme Funktionen zu programmieren. Diese haben keinen Namen, sondern der Code wird direkt hinter dem Schlüsselwort *function()* in geschweiften Klammern geschrieben.

Ein Sonderfall von Funktionen sind Methoden. Damit werden Funktionen bezeichnet, die an ein Objekt gebunden sind. Man kann sie aufrufen, in dem man den Namen des Objekts notiert, gefolgt von einem Punkt und danach den Namen der Methode mit Klammern.

Zusammenfassung

Dieses Kapitel beschäftigte sich mit Funktionen. Zunächst wurde der Begriff definiert und das Konzept dahinter dargelegt.

Weiterhin wurde die Syntax von Funktionen in JavaScript gezeigt.

Schließlich wurden einige Merkmale und Eigenschaften der JavaScript-Funktionen erläutert.

Aufgabe 6.1:

Wie ist eine normale Funktion in JavaScript aufgebaut?

Aufgabe 6.2:

Welche besonderen Arten von Funktionen gibt es in JavaScript?

Aufgabe 6.2:

Was sind Methoden in JavaScript?

7 JavaScript-Debugging in FireFox

Dieses Kapitel beschäftigt sich mit dem Debugging in JavaScript. Dabei wird zunächst das Ziel des Konzeptes Debugging beschrieben, danach geht es explizit um den Umgang und die Funktionsweise der Debug-Konsole in FireFox.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *den Sinn und die Vorteile von Debugging beschreiben zu können,*
- *den Aufbau der Debug-Konsole in FireFox kennen und*
- *die verschiedenen Methoden des Debuggings erklären und anwenden können.*

7.1 Ziel

Ein wichtiger Aufgabenbereich eines jeden Programmierers ist das Suchen und Finden von syntaktischen und logischen Fehlern im Code. In JavaScript führen Fehler oft nicht zum Absturz der Seite, sondern fallen erst spät auf, dafür umso schwerwiegender. Um die Fehler zu beheben und deren Ursachen ausfindig machen zu können, gibt es das Debugging. Alle modernen Browser besitzen einen eingebauten JavaScript-Debugger, hier soll es um den FireFox-Debugger gehen.

7.2 Aufbau

Um den Debugger in FireFox zu öffnen, wählt man im Menü von FireFox im Untermenü „Web-Entwickler“ den Punkt „Debugger“ aus. Es erscheint eine Toolbox im unteren Bereich des Browserfensters.

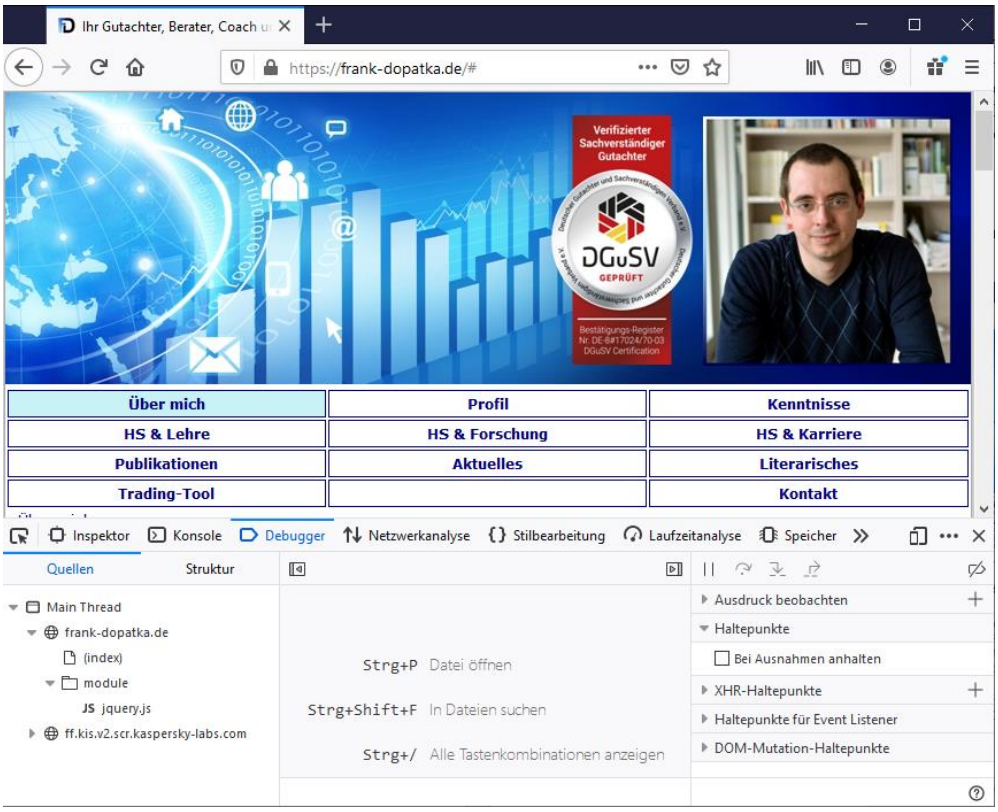

Abb. 7.1: Die Toolbox in FireFox (geöffnete Seite: https://frank-dopatka.de/#)

Die Toolbox öffnet sich direkt im Fenster „Debugger“. Die Benutzeroberfläche ist aufgeteilt in vier Bereiche. Am oberen Rand der Toolbox befindet sich die Werkzeugleiste. Darin kann zwischen den

verschiedenen eingebauten Werkzeugen des Browsers gewählt werden, hier liegt der Fokus jedoch nur auf dem Debugger. Das linke Fenster in der Toolbox ist eine Dateiliste aller JavaScript-Quellcodes auf der Seite. Man kann jede davon auswählen, um sie zu debuggen. Im mittleren Bereich wird der Code der aktuell ausgewählten Datei angezeigt. Rechts angeordnet ist die Variablenliste, in der die Variablen und ihre Werte an einer bestimmten Stelle im Code angezeigt werden.

7.3 Funktionsweise

Ohne die Möglichkeit des Debuggings ist die einzige Möglichkeit, Fehler im Code zu lokalisieren, nach und nach Blöcke im Code auszukommentieren, um herauszufinden, welche Teile des Codes nicht funktionieren. Mit *alert(...)* an den jeweiligen Stellen konnte man herausfinden, bis wohin das Programm ausgeführt wird. Um dies zu vereinfachen bietet der Debugger die folgenden Funktionen.

7.3.1 Debuggen mit Haltepunkten

Es besteht die Möglichkeit, in jeder Zeile des Quellcodes im mittleren Fenster einen Haltepunkt zu setzen, indem man auf die gewünschte Zeilennummer klickt. Der JavaScript-Code wird bei der Ausführung dann genau bis zu dieser Stelle ausgeführt. Es werden in der Variablenliste alle Variablen und Objekte mit deren Werten angezeigt. Diese können zusätzlich manuell verändert werden. Oberhalb der Variablen sind Navigierwerkzeuge, um sich dann im Code hin und herzubewegen, zu stoppen und weiterlaufen zu lassen.

7.3.2 Aufrufliste

Ein weiteres nützliches Feature ist die Aufrufliste (engl. „Call Stack“). Bei komplexeren Programmen kann es hilfreich sein, eine Übersicht über alle Funktionsaufrufe und deren Zusammenhänge zu haben. Dies bietet die Aufrufliste, die in der Variablenleiste als Menüpunkt angezeigt wird, wenn ein Haltepunkt gesetzt wurde und das Programm an diesem angekommen ist.

Zusammenfassung

In diesem Kapitel ging es um Debugging in JavaScript. Dabei wurde zunächst das grundlegende Konzept dahinter erläutert. Danach ging es explizit um die Debug-Konsole in FireFox, deren Aufbau und Funktionsweise.

Aufgabe 7.1:

Was ist das Ziel des Debuggings?

Aufgabe 7.2:

Welche Möglichkeiten gibt es in der FireFox-Konsole, JavaScript zu debuggen?

8 Der DOM-Baum

Dieses Kapitel beschäftigt sich mit dem Document Object Model. Zunächst wird dieses definiert und erläutert. Danach folgt ein kleiner Überblick über den Zugriff auf das DOM mithilfe von JavaScript, unterlegt mit einem Codebeispiel.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *den Begriff DOM beschreiben und erklären zu können,*
- *die Funktionsweise von JavaScript in Bezug auf das DOM zu kennen und*
- *einfache Funktionen zur dynamischen Veränderung des DOM mit JavaScript anwenden zu können.*

8.1 Begriff

Definition 8.1: DOM

Das Document Object Model (DOM) ist ein Standard des W3C. Er gibt vor, wie Elemente einer Webseite mithilfe einer Skriptsprache angesprochen, also ausgelesen oder verändert werden können. Hier bildet es die Schnittstelle zwischen dem HTML-Code (oder CSS) einer Webseite und JavaScript.

Wenn eine Seite über das Netzwerk empfangen wird, muss sie verarbeitet werden. Das ist die Aufgabe des Parsers, der den HTML-Code in die DOM-Struktur zerlegt. Konkret ist dies eine Baumstruktur, in der alle Objekte des Codes hierarchisch enthalten sind. Um Operationen auf der Webseite auszuführen wird dann nur noch diese Struktur verwendet, im Folgenden soll gezeigt werden wie dies in JavaScript konkret aussieht.

Es gibt unterschiedliche DOM-Versionen, hier soll es um das Level 2 gehen, da die älteren Versionen veraltet sind und von den Browsern nicht verwendet werden.

8.2 Zugriff mit JavaScript

Der DOM-Baum ist nicht statisch, sondern kann dynamisch verändert werden und somit eine Webseite dynamisch anpassen. Mit JavaScript können Methoden ausgeführt werden, also eine bestimmte Aktion, zum Beispiel um ein bestimmtes Element geliefert zu bekommen. Außerdem können Eigenschaften ausgelesen und verändert werden, also zum Beispiel der Inhalt eines HTML-Textfelds. Im Folgenden Beispiel wird dies gezeigt.

```
1  <html>
2      <body>
3          <p id="beispielAbsatz"></p>
4          <script>
5              document.getElementById("beispielAbsatz").innerHTML = "Änderung über DOM-Baum!";
6          </script>
7      </body>
8  </html>
```

Codebsp. 8.1: Zugriff auf den DOM-Baum

Im Beispiel wird mit HTML ein leerer Absatz mit der ID *beispielAbsatz* erstellt. Es folgt ein Stück JavaScript-Code, welcher zunächst mit Hilfe der Methode *getElementById(…)* das gewünschte Objekt, also den Absatz, geliefert bekommt. Die Methode wird an *document* aufgerufen, dieses Objekt beschreibt das aktuelle Fenster und ist das oberste Objekt in der Baumstruktur. An dem erhaltenen Element wird dann die Eigenschaft *innerHTML* verändert, also der Inhalt des Objekts. Somit enthält der Absatz bei Ausführung den Text „Änderung über DOM-Baum!“. Es ist üblich, an die Objekte mithilfe der ID des Namens zu kommen. Falls mehrere Objekte gleichzeitig angesprochen werden sollen, kann man das beispielsweise mit dem Tag-Namen oder dem Klassennamen machen.

Weitere Möglichkeiten der Modifikation von Elementen sind zum Beispiel *attribute*, um den Wert eines Attributes zu ändern oder *style.property*, um das Aussehen eines Elements zu verändern. Diese stünden dann im Code an der gleichen Stelle wie *innerHTML* im Codebeispiel 8.1.

Außerdem können Elemente zum Baum hinzugefügt (*document.createElement(…)*), gelöscht (*document.removeChild(…)*), angehängt (*document.appendChild(…)*) oder ersetzt (*document.replaceChild(…,…)*) werden. Um Text direkt in den Output-Stream von HTML zu schreiben wird *document.write(…)* verwendet.

Zusammenfassung

In diesem Kapitel ging es um das Document Object Model, wofür es gut ist und schließlich wie man es mit JavaScript modifizieren kann, um den HTML-Code von Webseiten dynamisch verändern zu können.

Aufgabe 8.1:

Aus was besteht das DOM und in welcher Struktur ist es aufgebaut?

Aufgabe 8.2:

Wie kann man auf ein Objekt mit einer bestimmten ID zugreifen?

Aufgabe 8.3:

Mit welcher Eigenschaft kann man den Inhalt eines Elements ändern?

9 Asynchrone Verarbeitung mit Event-Handling

Dieses Kapitel beschäftigt sich mit der Verarbeitung von Benutzereingaben und Benutzerverhalten mit Hilfe von Event-Handling. Dies wird an einem kleinen Beispiel erläutert.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *den Begriff Ajax definieren und erklären zu können,*
- *die Vorteile dieser Herangehensweise an die Webentwicklung zu verstehen und aufzeigen zu können und*
- *Programmcode schreiben zu können, mit dem man asynchron zum Browser mit JavaScript Benutzerverhalten verarbeitet und Anfragen an den Webserver stellt.*

9.1 Funktionsweise

Definition 9.1: Ajax

Asynchronous JavaScript And XML (Ajax) ist eine Herangehensweise an die Webentwicklung unter Verwendung bekannter Technologien wie JavaScript, XML und HTML. Dadurch wird es möglich, serverseitige Anfragen im Hintergrund ablaufen zu lassen, ohne dass der Benutzer etwas davon merkt.

Dieses Konzept ist also sozusagen die Grundlage der Abläufe auf Webseiten heutzutage, denn früher lagen Webseiten statisch vor, sie wurden vom Browser angefragt und geladen. Wenn die Seite angepasst werden sollte, musste sie komplett neu geladen werden.

Durch Ajax besteht dieses Problem nicht mehr. JavaScript tauscht im Hintergrund Daten mit dem Webserver aus, indem er eigene HTTP-Requests abschickt und die Server-Response verarbeitet. Das passiert asynchron zum eigentlichen Benutzer, somit wird der Browser nicht blockiert. Mit diesem System ist es möglich, viele neue Arten von Applikationen zu realisieren, wie zum Beispiel Kommunikationsprogramme oder Karten-Anwendungen, aber auch automatische Vervollständigung bei Suchmasken.

Um Webseiten dynamisch aufzubauen, zu modifizieren und zu ergänzen werden HTML-Elemente mit JavaScript-Code versehen, wie bereits in Kapitel 2.4.1 angesprochen. Dafür gibt es sehr viele Anwenderereignisse, die vom Benutzer durch Eingaben oder sein Verhalten ausgelöst werden können. Wenn ein entsprechender Event-Handler ausgeführt wird, manipuliert JavaScript das DOM der Seite.

9.2 Anwendung

Im folgenden Beispiel wird gezeigt, wie ein solcher Programmcode zur asynchronen Verarbeitung aussieht. Der Event-Handler kann entweder wie in Codebsp. 2.5 direkt in den Tag des Elements geschrieben werden, oder später mit *addEventListener* hinzugefügt werden.

```html
<html>
    <head></head>
    <body>
        <label for "kurs">Aktueller Bitcoin/USD Kurs:</label><br/>
        <textarea id="kurs" cols="10" rows="1">7500 USD</textarea><br/>
        <button type="button" id="aktualisieren" onclick="kursAktualisieren()">Kurs aktualisieren</button>
    </body>
</html>
```

Codebsp. 9.1: HTML-Code zum Ajax-Beispiel

Der Code erzeugt die folgende Seite:

Abb. 9.1: Erzeugte Seite zum Ajax-Beispiel

Der folgende JavaScript-Code verarbeitet die Benutzereingabe:

```
function kursAktualisieren(){
    var request = new XMLHttpRequest();
    request.open("GET", "kurs.txt");
    request.send();
    request.onreadystatechange = aktualisieren(){
      if(this.readyState == 4 && this.status == 200){
          document.getElementById("kurs").innerHTML = this.responseText;
      }
    }
}
```

Codebsp. 9.2: JavaScript-Code zum Ajax-Beispiel

Im Beispiel gibt es eine Textarea, in der der aktuelle Bitcoin/USD Kurs angezeigt werden soll. Mit dem Knopf unterhalb des Feldes ist es für den Benutzer möglich, diesen zu aktualisieren. Er besitzt einen EventHandler, bei einem Klick wird die JavaScript-Methode *kursAktualisieren()* aufgerufen. Diese erstellt zunächst einen neuen *XMLHttpRequest*. Diese sind die Hilfsmittel, mit denen JavaScript eigenständig Requests an einen Server stellen kann. Mit *open* wird der Request spezifiziert, es soll die GET-Methode verwendet werden und die Textdatei *kurs.txt* wird angefordert. Mit *send* wird der Request abgeschickt. Die Eigenschaft des Requests *onreadystatechange* ermöglicht es, eine Funktion auszuführen, sobald eine Response vorliegt. In diesem Fall ist das die Methode *aktualisieren()*, in dieser wird zunächst geprüft, ob der Response vollständig vorliegt und der Status OK ist. Falls ja wird der Text aus der Response in die Textarea *kurs* eingetragen, also dynamisch modifiziert. Zu beachten ist jedoch, dass XML nicht zwangsläufig das Übertragungsformat zwischen Client und Server sein muss, eine Alternative wäre zum Beispiel das JSON-Format.

Zusammenfassung

In diesem Kapitel ging es um die asynchrone Verarbeitung von Daten in JavaScript mit Zuhilfenahme des Event-Handlings.

Zunächst wurde die Grundidee von Ajax erläutert. Dann wurde der Ablauf einer asynchronen Verarbeitung von Benutzerverhalten beschrieben.

Schließlich wurde das Konzept anhand eines Beispiels aufgezeigt, in dem JavaScript auf den Klick des Benutzers reagiert, in dem es einen eigenen Request asynchron zum Browser an den Webserver sendet, um die erhaltenen Daten zur Ansicht des Benutzers anzuzeigen.

Aufgabe 9.1:

Wie funktioniert asynchrone Verarbeitung mit JavaScript?

Aufgabe 9.2:

Wie kann man mit JavaScript eigene Requests an den Webserver senden?

10 Ausblick auf JQuery

Dieses Kapitel soll einen kleinen Ausblick auf JQuery geben. Dafür wird zunächst das Projekt allgemein vorgestellt und die Vorteile erläutert. Danach werden die Möglichkeiten aufgezeigt, die sich damit auftun und die man damit umsetzen kann.

Sie sollten nach Durcharbeiten dieses Kapitels in der Lage sein,

- *die Idee hinter dem Projekt JQuery darlegen zu können und*
- *die Möglichkeiten aufzuzeigen, die dem Entwickler mit JQuery geboten werden.*

10.1 Begriff

JQuery ist eine JavaScript Bibliothek, die den Umgang mit JavaScript, vor allem in Bezug auf das DOM, erleichtern soll. Frei nach dem Motto des Open-Source Projekts „write less, do more“, stellt es viele Möglichkeiten zur Verfügung, die mit wenigen Zeilen Quellcode umgesetzt werden können. Außerdem ermöglicht JQuery technische Unterschiede zwischen verschiedenen Browsern zu überbrücken, indem diese intern behandelt werden. Somit ist der Entwickler nur noch dafür zuständig, das allgemeingültige Programm zu schreiben.

Um JQuery in JavaScript einzubinden, stellt man es entweder auf dem eigenen Server bereit und kann es dann über den Ort, an dem es liegt, adressieren, oder man bindet es von einem *Content Delivery Network* wie Google oder Microsoft ein, welche dieses zentral bereitstellen.

10.2 Möglichkeiten

Mit JQuery wird es viel einfacher, auf einzelne HTML-Elemente einer Seite über das DOM zuzugreifen. Was mit JavaScript nur umständlich mit *getElementById* usw. ging, kann nun sehr kompakt über die Funktion $ erfolgen. Dies ist auch die einzige Funktion, die für den Zugriff auf Elemente verwendet wird. Es liefert das oder die passenden Elemente immer in einem Array-Objekt. Kurz gesagt erleichtert es also die DOM-Manipulation mit JavaScript.

Genauso kompakt können die gelieferten Elemente dann geändert werden, also zum Beispiel, um deren Eigenschaften oder Werte zu modifizieren. Dafür verwendet man standardmäßig *attr()* mit einem Parameter, dem Namen der Eigenschaft, um den Wert auszulesen (Getter) oder mit zwei Parametern, wobei der zweite der neue Wert ist, der eingetragen werden soll (Setter).

JQuery lässt sich auch leicht mit dem Event-Handling verbinden. Dadurch ist es möglich, noch leichter auf Benutzerverhalten zu reagieren, zum Beispiel dynamische Elemente oder Effekte in die Webseite mit einzubauen.

Es ist weiterhin möglich, gleichzeitig Ajax mit in das Programm einzubauen und somit die Verwendung von JQuery zusätzlich mit Serveranfragen zu unterstützen. Genauso gibt es noch viele weitere Plug-Ins, die man einfach einbinden und zusammen verwenden kann, um es als Entwickler möglichst einfach und kompakt zu haben.

Zusammenfassung

Dieses Kapitel sollte einen kurzen Ausblick auf das Konzept von JQuery geben. Dafür wurde zunächst die Grundidee dieser Open-Source Technologie erläutert. Zuletzt wurden die verschiedenen Möglichkeiten aufgezeigt, die die Verwendung von JQuery bietet, um den Entwickler dabei zu unterstützen, seine Programme möglichst kompakt und zielorientiert zu schreiben.

Aufgabe 10.1:

Was ist die Grundidee von JQuery?

Aufgabe 10.2:

Nennen sie drei Möglichkeiten, die JQuery für den Entwickler bietet.

A. Lösungen der Aufgaben zur Selbstüberprüfung

1.1

Das Einsatzgebiet der JavaScript Programmierung ist in erster Linie die webbezogene Programmierung, wobei diese meist clientseitig implementiert ist.

1.2

Das ECMAScript standardisiert ein JavaScript System, um verschiedene Implementationen einheitlich zu halten. Es legt unter anderem die Sprachelemente und Sicherheitsprinzipien fest. Die bekannteste Implementierung ist JavaScript.

1.3

var beispielZahl = 5;

2.1

JavaScript kann direkt in den HTML-Code in ein <*script*> Tag geschrieben werden, aus einer externen Datei eingebunden werden, in einen Link eingebaut werden oder in einem Event-Handler hinterlegt werden.

2.2

Zum Einbinden in den HTML-Code muss der JavaScript-Code zwischen den Tags <*script*> und <*/script*> stehen.

2.3

Um ein externes Skript einzubinden, muss dieses im <*script*> Tag mit href="*name*.js" referenziert werden.

2.4

Es wird das Protokoll *javascript:* benötigt, hierauf folgen die Anweisungen als JavaScript-Code.

2.5

Um auf das Benutzerverhalten einzugehen, werden in JavaScript EventHandler benutzt. Der hinterlegte Code wird dann ausgeführt, sobald das jeweilige Ereignis eintrifft.

3.1

In JavaScript kann man eine Variable mit den Schlüsselworten *var*, *let* und *const* (Konstante) deklarieren.

3.2

Variablen in JavaScript sind dynamisch typisiert, das heißt eine Variable hat den Datentypen des Wertes, der ihr zugewiesen wird, und zwar erst nach der Wertezuweisung. Somit kann sich der Datentyp bei Veränderung der Variablen mit der Zeit ändern.

3.3

In JavaScript gibt es den lokalen und den globalen Scope. Jede Funktion erschafft einen lokalen Scope. Falls eine Variable in einer Funktion deklariert wird, ist sie nur in deren lokalen Scope. Alle anderen Variablen haben den globalen Scope, es kann von überall aus auf sie zugegriffen werden.

3.4

Unter dem Begriff Hoisting versteht man in JavaScript die Eigenschaft des Interpreters, alle Variablendeklarationen an den Anfang des jeweiligen Sichtbarkeitsbereichs zu verschieben.

3.5

Die Truthiness legt fest, welche Werte zu welchen Wahrheitswerten von JavaScript ausgewertet werden.

3.6

In JavaScript werden alle Werte, wenn nötigt, implizit zu den benötigten Datentypen konvertiert, wobei dies immer zum größeren Typ hin erfolgt. Man kann Datentypen jedoch auch manuell konvertieren.

4.1

In JavaScript gibt es die *if*-Anweisung mit den besonderen Ausprägungen *if/else* und *if/else if*. Außerdem gibt es die *switch* Anweisung.

4.2

In JavaScript gibt es *while*, *do…while*, *for*, *for…in* und *for…of* Schleifen.

5.1

var autos = [“BMW“, “Mercedes“];

5.2

Mit der Methode *length*.

6.1

Eine Funktion beginnt mit dem Schlüsselwort *function*, gefolgt von dem Namen der Funktion, Klammern, in denen optional Parameter festgelegt werden können und zuletzt geschweifte Klammern, in denen der auszuführende Code steht.

6.2

In JavaScript kann man unter anderem Funktionen innerhalb von Funktionen deklarieren. Weiterhin gibt es die Möglichkeit, anonyme Funktionen zu schreiben.

6.3

Methoden sind in JavaScript Funktionen, die an ein Objekt gebunden sind.

7.1

Das Ziel des Debuggings ist das Finden von Fehlern im Quellcode, um diese zu beheben.

7.2

In der Debug-Konsole von FireFox ist es möglich, Haltepunkte zu setzen, an denen das Programm angehalten wird. An diesen werden dann alle aktuellen Variablen und Werte angezeigt. Außerdem gibt es die Möglichkeit, sich die Aufrufliste aller Funktionen anzeigen zu lassen.

8.1

Das Document Object Model besteht aus allen Objekten der HTML-Seite, die es repräsentiert. Diese sind in einer Baumstruktur aufgebaut.

8.2

Mit *getElementById(…)*.

8.3

Mit *innerHTML(…)*.

9.1

JavaScript ist in der Lage, eigenständig mit dem Webserver zu kommunizieren. Das bedeutet, man kann mit JavaScript Webseiten dynamisch aufbauen und modifizieren. Dies passiert asynchron zum Browser. Er wird nicht blockiert, sondern JavaScript verändert das DOM der Seite einfach, sobald die gewünschten Ergebnisse vom Server vorliegen.

9.2

Um Requests an den Webserver zu senden, erstellt man ein neues Objekt des Typs *XMLHttpRequest*. An diesem wird mit *open* die Anfrage spezifiziert und mit *send* abgeschickt.

10.1

Die Grundidee hinter JQuery ist es, dem JavaScript-Entwickler einen einfacheren Umgang mit dem DOM zu ermöglichen, um es kompakt und übersichtlich modifizieren und so Webseiten einfach dynamisch verändern und erweitern zu können.

10.2

Man kann mit der Funktion *$* einfach an alle Elemente im DOM gelangen, diese dann mit *attr* in ihren Eigenschaften verändern und auf Benutzerverhalten leicht reagieren.

B. Literaturverzeichnis

[SK09] Stefan Koch: JavaScript: Einführung, Programmierung und Referenz – inklusive Ajax; dpunkt.verlag; 2009; ISBN 978-89864-594-2

[SS11] Steve Suehring: JavaScript – Schritt für Schritt; Microsoft Press Deutschland; 2011; ISBN 978-3-86645-551-1

[MH12] Marijn Haverbeke: Die Kunst der JavaScript – Programmierung; dpunkt.verlag; 2012; ISBN 978-3-89864-787-8

[CW10] Christian Wenz: JavaScript und AJAX; Rheinwerk Computing; 7. Auflage 2010; ISBN 3-89842-859-1

C. Abbildungsverzeichnis

D. Codebeispielverzeichnis