edv-buchversand.de

JavaScript Schnelleinstieg

Dynamische Webseiten programmieren in 14 Tagen

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

1 Vorbereitungen und Werkzeuge

1.1 HTML und CSS

1.1.1 Korrektes HTML

Es ist nicht wichtig, ob Sie bereits PHP, TypeScript oder Java programmiert haben: Ohne solide Kenntnis und Erfahrung mit HTML und CSS ist der Einstieg in JavaScript steil und steinig. Schließlich ist ein Hauptgrund für den Einsatz von JavaScript der Eingriff in die Darstellung der Webseite. Valides – also korrektes – HTML ist die Basis für JavaScript. Obwohl die Browser viele HTML-Fehler ignorieren, schlagen falsch verschachteltes HTML und fehlende Hochkommas im HTML-Tag zurück. Darum gehört der Validator zum Alltag des JavaScript-Programmierers.

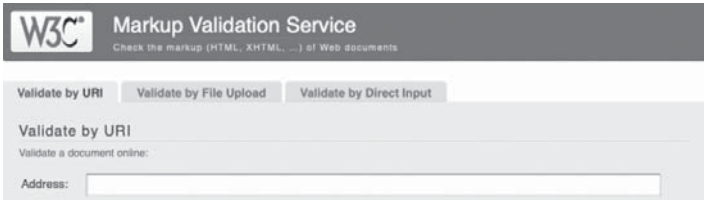


Abb. 1.1: *https://validator.w3.org* – Prüfen, ob die HTML-Struktur stimmt

1.1.2 Wie viel CSS-Kenntnis ist wichtig?

Wie gut Sie sich mit CSS auskennen sollten, hängt davon ab, auf welche Einsatzgebiete von JavaScript Sie sich in erster Linie vorbereiten:

- Animationen und Effekte
- Design und Layout von Webseiten
- Kommunikation mit dem Server

Wenn Sie JavaScript einsetzen wollen, um die Grenzen von CSS-Animationen zu überschreiten – etwa Animationen starten, wenn sie im Browserfenster sichtbar werden –, sind gute Kenntnisse in CSS-Animationen gefragt.

Bei Änderungen am Design von Content-Management-Systemen wie WordPress, Joomla oder Drupal kann das CSS anspruchsvoll sein. Dort sitzen auf einem einzelnen Element gut ein Dutzend CSS-Eigenschaften aus unterschiedlichen Quellen.

Konzepte wie `Cascading` (das Einfließen von Stilen aus vorangegangenen CSS-Regeln) und CSS-Selektoren brauchen ein tiefes Verständnis. Ein `section`-Element einer Seite beherbergt oft ein gutes Dutzend Stile aus verschiedenen Quellen:

Abb. 1.2: Der Erste-Hilfe-Koffer für den Einblick ins CSS ist die Browser-Konsole.

Die Kommunikation mit Anwendungen auf dem Server ist ein weiteres Einsatzgebiet von JavaScript. Das Einlesen und Weiterreichen von Benutzereingaben kommt mit einfachen Eingriffen in die Darstellung im Browser aus. Hier reichen solide Grundlagen von CSS.

1.1.3 HTML- und CSS-Kenntnisse – alles halb so wild

Zur Programmierung mit Sprachen wie Java oder C gehört die Formatierung der Ausgabe. Die Benutzerschnittstelle einer Desktop-Anwendung ist ein Monster von Menüs und Programmfenstern sowie die Reaktion auf Benutzereingaben und auf Nachrichten vom Betriebssystem. In JavaScript überneh-

men HTML und CSS die Formatierung der Benutzerschnittstelle vollständig. Selbst wenn die Kenntnis von HTML und CSS nur oberflächlich ist, haben Sie damit schon einen großen Teil des Weges zur Programmierung von Webseiten »im Kasten«.

1.2 JavaScript einbinden

1.2.1 Ein Sandkasten für JavaScript – Sandbox

JavaScript braucht keinen Webserver für die Ausführung, sondern läuft in einem beliebigen Ordner auf dem eigenen Rechner. Ein Texteditor und ein Browser, das ist schon alles, was für die ersten Schritte mit JavaScript erforderlich ist.

Dem Editor (besser »Programmeditor«) ist ein eigener Abschnitt gewidmet, aber für die ersten Schritte reichen etwa TextEdit auf dem Mac, Notepad unter Windows oder nano unter Unix. Wichtig ist, dass der Editor keine Formatierung einbringt – keine Buttons für fette oder kursive Schrift, keine Schriftgrößen oder -farben.

Die Vorbereitungen für die ersten Schritte mit JavaScript beschränken sich also auf das Anlegen eines Ordners – der Sandbox. Der Name des Ordners ist nicht relevant – `meine-webseite`, `sandbox` oder `klara`. JavaScript wird im Browser ausgeführt, für die Formatierung der Ausgabe sorgen HTML und CSS. Das macht den Einstieg in JavaScript einfacher als das Erlernen anderer Programmiersprachen.

```
▼ sandbox
  ▶ index.html
  ▶ script.js
```

Im ersten Teil dieses Buchs ist die HTML-Datei nur ein Grundgerüst aus `head` und `body`.

HTML:

```
<head>
   <title></title>
</head>
<body>
   <h1>Hallo Browser!<h1>
</body>
```

Die Datei `index.html` öffnet sich durch einen Doppelklick im Browser, aber Änderungen nach dem Öffnen der Seite zeigt der Browser erst an, wenn die Seite neu geladen wird. Darum empfiehlt Abschnitt 1.4 Programmeditoren mit einem Live-Server. Doch zunächst erst einmal zum Einbinden von JavaScript in Webseiten.

1.2.2 Das HTML-script-Element

JavaScript-Anweisungen können mit dem HTML-script-Tag direkt in das HTML-Dokument gesetzt oder als Datei mit der Namenserweiterung `.js` in eine gesonderte Datei geschrieben werden. Das script-Tag bindet die externe JavaScript-Datei mit einem src-Attribut ein.

Das script-Element kann an beliebiger Stelle eines HTML-Dokuments stehen: im Seitenkopf oder innerhalb der öffnenden und schließenden body-Tags. Die empfohlene Arbeitsweise ist eine Skript-Datei mit der Dateierweiterung `.js` und dem defer-Attribut nach dem Laden der CSS-Dateien im Kopf der HTML-Datei.

HTML:

```
<head>
   <script src="script.js" defer></script>
</head>
<body> … </body>
</html>
```

Viele Content-Management-Systeme binden das JavaScript noch vor dem schließenden body-Tag ein. Ein defer-Attribut wird dann nicht gebraucht.

```
<head> … <head>
<body>
   …
   <script src="script.js"></script>
</body>
```

Am Ende können Sie kleine Skripte auch direkt vor dem schließenden body-Tag einsetzen.

```
<body>
   <script>
      Anweisung;
      Anweisung;
   </script>
</body>
```

Im Allgemeinen ist es nicht empfehlenswert, die Skript-Anweisungen direkt ins HTML zu setzen. Aber wenn das Programm nur wenige Zeilen enthält und nur auf einer Seite gebraucht wird, spricht nichts dagegen.

1.2.3 JavaScript-Module: Import und Export

Mit ECMAScript 6 ist eine weitere Methode zum Einbinden von Skript-Dateien angekommen, die heute von allen modernen Browsern unterstützt wird: JavaScript-Module. Wenn Sie Skripte von Dritten einbinden, wird ihnen diese Option früher oder später begegnen.

Module organisieren Skript-Dateien, indem sie Anwendungen in überschaubarere Teile herunterbrechen und sie in separaten Dateien speichern. Das »Hauptprogramm« holt diese Fragmente in das Skript zurück: Sie werden als Module importiert.

Das type-Attribut des script-Elements kennzeichnet JavaScript-Dateien als `module`. Sowohl Skript-Dateien, die Module importieren, als auch Skripte, die Elemente exportieren, müssen als `type="module"` notiert werden.

```
<script src="app.js" type="module" defer></script>
```

Code, der in der Anwendung nur stören würde, wird in ein Modul exportiert – z.B. tools.js. Die zentrale Skript-Datei importiert die Module, die tatsächlich gebraucht werden.

Im Rahmen dieses Buchs bleiben wir beim klassischen Einbinden von JavaScript-Dateien mit dem script-Tag.

1.3 JavaScript laden

1.3.1 Ausführungsreihenfolge beim Laden von Webseiten

Der Browser liest das HTML-Dokument Zeile für Zeile von oben nach unten und beginnt bereits mit der Darstellung der Seite, bevor er das Dokument

vollständig geladen hat. Bei einer externen Skript-Datei stoppt der Browser, bis die Datei heruntergeladen ist. Das kann zu einer spürbaren Latenz der Anzeige führen. Große Skript-Dateien im Seitenkopf führen zu einer Verzögerung im Aufbau des Dokuments, dem »Rendern«.

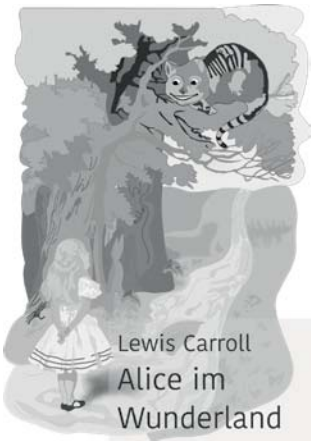
Name	Status	Type	Initia…	Size	Time	Waterfall
index.html	200	document	Other	4.0 kB	44 ms	
style.css	200	stylesheet	inde…	4.2 kB	21 ms	
logo.svg	200	svg+xml	inde…	736 B	40 ms	
a.jpg	200	jpeg	inde…	20.2 kB	56 ms	
kaktus.jpg	200	jpeg	inde…	33.0 kB	39 ms	
clock.jpg	200	jpeg	inde…	20.4 kB	72 ms	
stamp.jpg	200	jpeg	inde…	22.5 kB	60 ms	
pack.jpg	200	jpeg	inde…	18.0 kB	73 ms	
scripts.js	200	script	inde…	1.1 kB	73 ms	
CC-BY-SA.svg	200	svg+xml	inde…	1.9 kB	73 ms	

Abb. 1.3: Der »Wasserfall« stellt das Laden der Elemente der Webseite als Grafik dar.

1.3.2 Ausführungsreihenfolge festlegen mit async und defer

Skripte wurden bis vor wenigen Jahren häufig am Ende des Dokuments vor dem schließenden body-Tag geladen, um ohne großen Aufwand sicherzustellen, dass alle Ressourcen (Bilder, CSS-Dateien …) bereits geladen sind. Heute hat das script-Tag zwei Attribute: `defer` oder `async`. Sie verhindern den Zwischenhalt, aber laden die Skript-Datei »just in time«.

Mit `defer` und `async` können Sie Skript-Dateien im head-Element des Dokuments einbinden und sicherstellen, dass der Browser sie in der richtigen Reihenfolge lädt, ohne den Seitenaufbau zu unterbrechen.



```
<script src="script1.js"></script>
<script src="script2.js"></script>
```

`script1.js` lädt ein Bild, `script2.js` wendet einen Filter auf das Bild an. Sitzen die Skript-Tags im Kopf der Seite, erzeugen sie beide Fehler: `script1.js` kennt das HTML-Element für die Platzierung des Bilds nicht, `script2.js` kann den Filter nicht auf das Bild anwenden.

```
<script src="script1.js" defer></script>
<script src="script2.js"></script>
```

Mit `defer` wird das Bild zwar geladen und eingesetzt, aber dieser Prozess dauert an, während der Browser `script2.js` schon ausführt. Zum Zeitpunkt der Ausführung von `script2.js` ist das Bild noch nicht geladen. Dieselbe Situation tritt ein, wenn `script1.js` als `async` markiert wäre.

```
<script src="script1.js" defer></script>
<script src="script2.js" defer></script>
```

Der Browser lädt Skripte mit einem defer-Attribut in der Reihenfolge, in der sie im Code erscheinen. Dabei wird der Browser nicht blockiert, er führt diese Skripte aber erst aus, wenn das Dokument vollständig geladen ist.

Skripte mit einem async-Attribut werden ebenfalls ohne eine Blockade ausgeführt, aber ohne Beachtung der Reihenfolge. Das kann dazu führen, dass `script2.js` vor `script1.js` geladen und ausgeführt wird. Wenn es nicht auf die Reihenfolge beim Laden der Skripte ankommt, wäre `async` die praktikable Lösung.

1.4 Mein Freund, der Editor

1.4.1 Die Werkzeuge der Programmeditoren

Sie können jeden Texteditor benutzen, aber gute Programmeditoren sind schon für HTML und CSS unersetzlich, für JavaScript gilt dies umso mehr. Die Anforderungen sind:

- **Syntax-Highlighting** – der Editor macht die Grammatik der Programmiersprache durch unterschiedliche Farben sichtbar.

- **Einrücken** – bringt die hierarchische Struktur von Code-Blöcken sauber zum Vorschein und erleichtert die Fehlersuche.
- **Code Folding** (Blöcke zusammenfalten) – verbirgt Code-Segmente, die aktuell nicht gebraucht werden.
- **Code Completing** – zeigt nach wenigen Buchstaben Hinweise auf Befehle, Anweisungen und Methoden.
- **Automatisches Klammern** – setzt direkt doppelte Hochkommas, runde, eckige und geschweifte Klammern und positioniert den Cursor in die Klammer.
- **SplitView** oder **SplitScreen** – zeigt ein Dokument in einem geteilten Fenster, um einen gleichzeitigen Einblick in lange Codeseiten zu bieten.

1.4.2 Visual Studio Code

Der beste Editor ist der, den man durch und durch kennt. Tatsächlich finden Sie heute viele hervorragende kostenlose Code-Editoren im Netz.

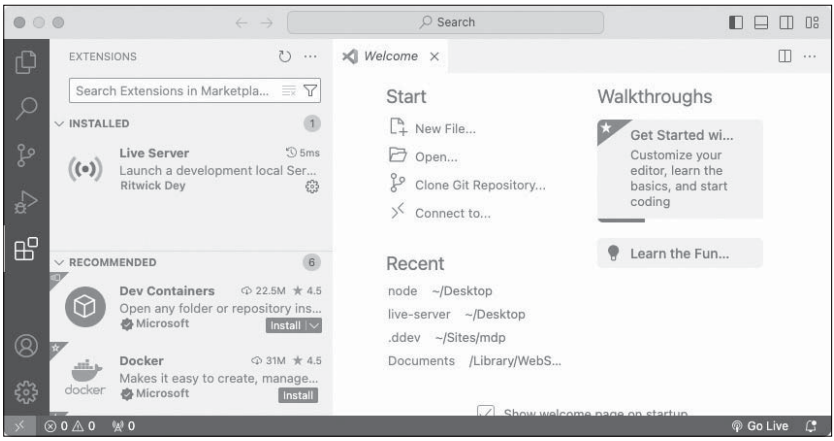


Abb. 1.4: Screenshot: Visual Studio Code – empfohlene Erweiterung: Live Server

Visual Studio Code (VSC) von Microsoft ist der Industriestandard und läuft »auf 'nem Bierdeckel« (charmante Umschreibung für »unter jedem Betriebssystem«). Anforderungen wie

- Syntax-Highlighting,
- das Zusammenklappen von Anweisungsblöcken,
- Auto-Vervollständigen

erfüllen die meisten Programmeditoren aus dem Effeff, aber Visual Studio Code kann mehr. Für die JavaScript-Entwicklung ist eine Live-Server-Umgebung unbezahlbar. Ein Live-Server ist eine Anwendung im Code-Editor, die sich zum Webserver aufspannt. Sie sehen den Effekt von Änderungen nach dem Speichern, ohne die Seite im Browser durch ein Refresh neu zu laden. Für Visual Studio Code (VSC) gibt es diese Funktion als Erweiterung.

1.4.3 Bare Bones BBEdit

BBEdit läuft nur unter macOS und kostet etwa 50 €, aber hat den Live-Server schon an Bord, ebenso Syntax-Highlighting, Code Folding, Code Completing, automatische Klammern und SplitView. Ohne eine Erweiterung oder ein Plug-in lädt BBEdit Dateien ohne FTP/SFTP-Programm direkt vom Webserver und beim Speichern wieder zurück auf den Webspace.

1.4.4 IDE – Integrierte Entwicklungsumgebung

Eine IDE (Integrated Development Environment) ist eine ganze Suite mit Entwicklerwerkzeugen, die mehrere Programmiersprachen unterstützt und häufig von großen Teams genutzt wird. Meist macht die Konfiguration viel Arbeit und überfordert Einsteiger schnell ohne jeglichen Mehrwert. Beispiele für IDEs sind PhpStorm und Xcode für macOS.

1.5 Die Browser-Konsole – die Sprechstundenhilfe

1.5.1 Browser

Moderne Browser sind die Grundlage für das Design und Layout von Webseiten mit HTML und CSS. Selbst heute, da sich die Browser nach den Zwistigkeiten der Frühzeit einander angenähert haben, reicht der bevorzugte Browser nicht, um alle JavaScript-Programme ausreichend zu testen. Kleine Abweichungen gibt es immer noch, und am besten versorgen Sie sich mit allen Browsern, die auf dem eigenen System verfügbar sind. Auch ältere Version der Browser sollten in der Sammlung nicht fehlen.

Der Blick auf den Desktop-Monitor reicht nicht. Mobile Geräte für Android und iOS werden im Laufe der Entwicklung zu einer unverzichtbaren Ergänzung beim Testen Ihrer Anwendungen. Während es für macOS Simulatoren sowohl für iOS-Geräte als auch für Android gibt, bleibt PC- und Unix-

Benutzern die Sicht auf iOS verwehrt, wenn sich keine hilfreiche Freundin mit einem iPhone findet.

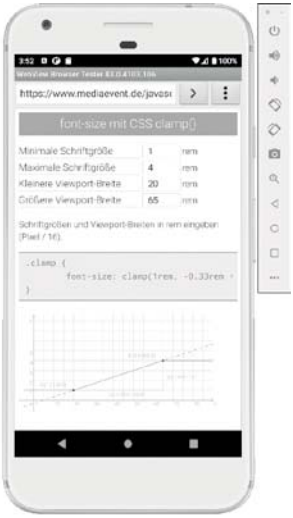


Abb. 1.5: Android-Simulator

Abb. 1.6: iOS-Simulator

Die Installation der Simulatoren ist kein »Klick mich, hier bin ich«, aber der Aufwand der Installation lohnt sich. Die Darstellung der Simulatoren ist exakter, beim Testen sind sie schneller bei der Hand als eine Sammlung mobiler Geräte.

Apple hat den Simulator für mobile Geräte von der Apple Watch bis zum iPad in die Entwicklungsumgebung `Xcode` unter `Developer` > `Tools/Simulator` eingebettet. Apple-Nutzer können Xcode kostenlos im App Store laden.

Für Android-Geräte stellt Google Android Studio auf *https://developer.android.com/studio* für Windows und macOS zur Verfügung.

1.5.2 Die Browser-Konsole

Das wichtigste Werkzeug der Browser wiederum ist ihre Browser-Konsole, die sich hinter den Entwicklerwerkzeugen verbirgt. Die Konsole zeigt Fehler an, hilft bei der Suche nach Logikfehlern und ist die erste Wahl, um eine Anweisung ohne großen Aufwand zu testen.

Alle Browser – von Chrome über Edge und Firefox bis zu Safari – haben eine Konsole, aber der Zugriff auf die Konsole ist bei jedem Browser unterschiedlich.

1.5.3 Safari

Der Apple-Browser eröffnet den Weg zur Browser-Konsole erst, wenn unter den Safari-Einstellungen die Entwicklerwerkzeuge aktiviert werden.

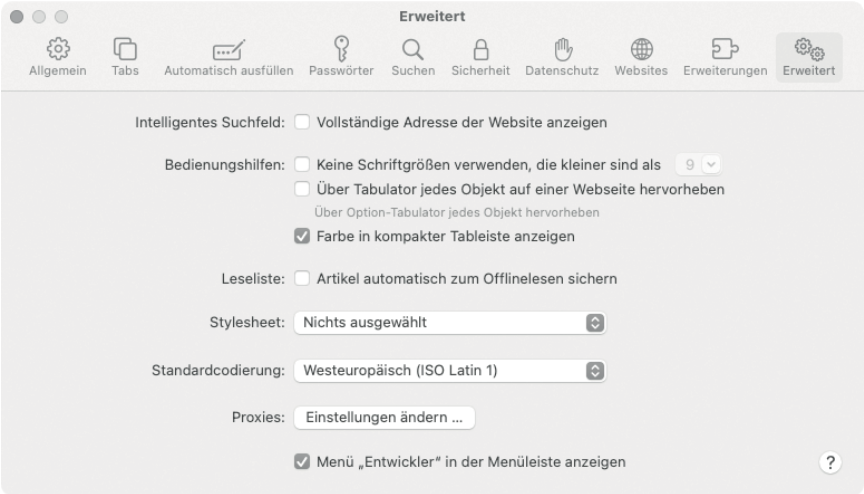


Abb. 1.7: Safari (macOS): Einstellungen – Erweitert – Menü »Entwickler« in der Menüleiste anzeigen

Von da an sitzt der Punkt »Entwickler« dauerhaft und gut sichtbar in der Menüleiste.

1.5.4 Chrome

Safari und Chrome haben denselben Ursprung – Webkit –, aber die Konsole liegt bei Chrome hinter den drei Punkten ⋮ rechts oben im Browserfenster.

Abb. 1.8: Unter »Weitere Tools« liegen die »Entwicklertools« weit unten.

1.5.5 Firefox

Firefox zeigt auf der rechten Seite des Browserfensters die drei kleinen Balken eines Hamburger-Menüs ≡, von dort aus am unteren Ende des Menüs »Weitere Werkzeuge« >> »Browser-Konsole«.

1.5.6 Ein Blick in die Browser-Konsole

Trotz kleiner Unterschiede in der Funktionalität agieren die Browser-Konsolen aller Hersteller in gleicher Weise. Sie zeigen uns den HTML-Quelltext der Seite, aber für die JavaScript-Entwicklung ist das Register `Console` der Dreh- und Angelpunkt.

JavaScript zeigt mit dem Befehl `console.log` Testausgaben, die sicherstellen, dass das Programm auf dem richtigen Weg ist, zeigt Fehlermeldungen und die Ursache des Fehlers (Abbildung 1.9).

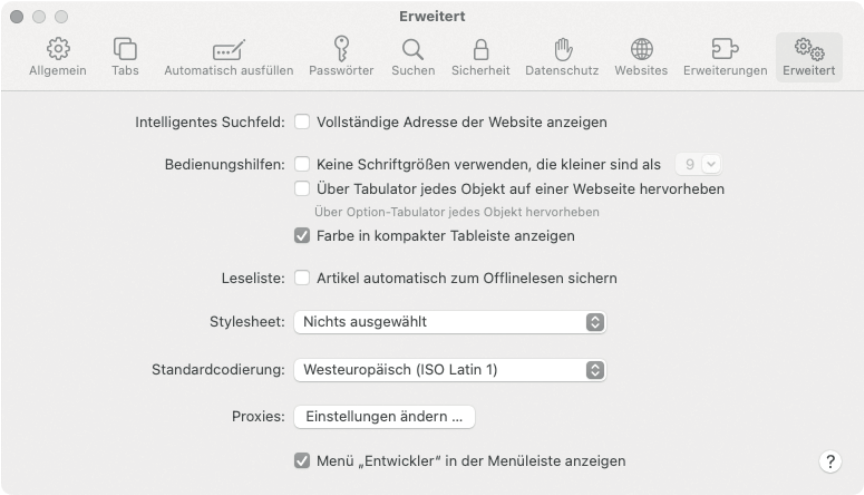


Abb. 1.9: Die Console zeigt Testausgaben, die aktuellen Werte von Variablen und das Ergebnis einfacher Abfragen.

Viele der Beispiele in diesem Buch zeigen Konsolen-Ausgaben. Um sie von Code-Beispielen zu unterscheiden, haben sie ihr eigenes Format:

1.6 Praktische Konsolen-Befehle

1.6.1 Die Konsole unter den Entwicklertools

Ohne die Konsole der Browser ist die JavaScript-Entwicklung ein Blindflug. Das ist nun mal so: Die meiste Zeit beim Programmieren verbringen wir mit der Suche nach Fehlern.

Es gibt zwei grundsätzliche Wege, mit der Konsole zu arbeiten:

- entweder durch `console.log()`-Anweisungen innerhalb des JavaScript-Codes,
- oder indem man Anweisungen direkt in die letzte Zeile der Konsole schreibt.

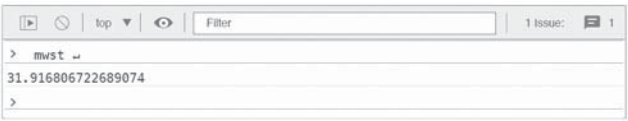
Die Browser-Konsolen – ob Chrome oder Firefox, Safari oder Edge – unterscheiden sich nur marginal.

1.6.2 Direkte Eingaben in der Konsole

Direkte Eingaben nach dem ›-Zeichen der untersten Zeile der Konsole testen Variablen und Objekte und zeigen ihre aktuelle Struktur. Öffnen Sie die Browser-Konsole einer beliebigen Webseite und schreiben Sie die folgende Anweisung hinter das ›-Zeichen:

`alert()` öffnet ein Dialogfenster und zeigt den Text »Hallo Welt!« an, obwohl die Anweisung im Skript nicht vorkommt. Sie können Anweisungen ausführen und testen, bevor Sie Änderungen im Skript vornehmen. Die folgenden Skript-Anweisungen berechnen die Mehrwertsteuer aus einem Bruttobetrag:

HTML:

```
<!DOCTYPE html>
<html lang="de">
<head>
   <title>Mehrwertsteuer berechnen</title>
</head>
<body>
<script>
const x = 199.90;
const mwstSatz = 19;
const mwst = x * mwstSatz / 119;
</script>
</body>
</html>
```

Mal eben in der Konsole überschlagen, ob die Formel so richtig ist: