JavaScript 1

Einstieg

JavaScript

September 1995
LiveScript
Netscape Navigator 2.0

August 1996
JScript
Microsoft Explorer

Oktober 1998
ECMA-262

Juni 2015
ECMA-262
ECMAScript 2015
„ECMAScript 6“
JavaScript ES6



Dezember 1995
JavaScript 1.0
Kooperation
Netscape & Sun

April 1996
JavaScript 1.1
Navigator 3.0

Bis 1994:

European
Computer
Manufacturers
Association

danach: Ecma international

ECMAScript und JavaScript werden in der Praxis synonym verwendet. ECMAScript wird jährlich aktualisiert und erweitert

„JavaScript“ ist eine Marke von Sun, heute von Oracle

JavaScript hat mit Java *nichts* zu tun!

Scriptsprache

„Scripte“ sind ursprünglich einfache Ergänzungen zu Programmen oder anderen Sprachen (Makros)

Heute: Ausgewachsene, leistungsfähige Programmiersprachen

Scripte werden erst beim Aufruf ausgeführt und „on the fly“ in Maschinencode übersetzt (Interpretersprachen)

Benötigen eine Laufzeitumgebung (= den Interpreter)

Der Webbrowser besitzt eine JavaScript-Engine, die den Code ausführt

Jeder Browser hat seine eigene Engine

Allgemeine Eigenschaften

JavaScript ist eine dynamisch typisierende, clientseitige Scriptsprache

Wird in der Regel in HTML eingebettet und vom Browser ausgeführt:

- Quellcode wird im Client verarbeitet und ist dort sichtbar
- Ausführung von JavaScript kann im Client blockiert werden

Ziele z.B.: Manipulation des DOM, Interaktion, Datenvalidierung …

Läuft in einer „Sandbox“

Kein – oder nur sehr eingeschränkter – Zugriff auf

- Dateisystem
- Hardware
- Datenbanken

Es gibt wie immer Ausnahmen …

Alles bisher Gesagte gilt nur für Plain Vanilla JavaScript. Es gibt Ausnahmen („JavaScript auf dem Server“):

- **V8:** Laufzeitumgebung für JavaScript; Rendering-Engine von Google Chrome, aber auch ohne Browser benutzbar
- **Node.js:** Basiert auf V8 und ist ebenfalls eine Laufzeitumgebung, die unabhängig von einem Browser betrieben werden kann
- **Electron:** Basiert auf Node.js und ist ein Framework für plattformunabhängige Anwendungen. Beispiele:
  - Teams
  - Visual Studio Code
  - Diagrams.net (= draw.io)

JavaScript & HTML

JavaScript wird (in der Regel) mit HTML kombiniert

Script-Blöcke können (fast) an beliebigen Stellen eingefügt werden (Head & Body)

Alle Blöcke bilden *ein* großes JavaScript-Programm

Die Anweisungen werden in der Reihenfolge ausgeführt, in der sie im Quellcode stehen

Es gibt vor allem zwei Möglichkeiten, HTML und JavaScript zu kombinieren:

- Externes Script
- Interne Script-Anweisungen

Externes Script

Über `<script src="…"></script>` an beliebigen Stellen

`<script>`-Tag muss geschlossen werden!

Steht traditionell im `<head>`-Bereich

Heute: Besser ans Ende setzen (vor `</body>`)

```
<html>
  <head>
    <title>Test</title>
    <meta charset = "utf-8">
    <script src="scriptEins.js"></script>
  <head>
  <body>
   …
    <script src="scriptZwei.js"></script>
  </body>
</html>
```

Internes Script

An beliebigen Stellen (<head> & <body>):

```
<!-- beliebiges HTML -->
<script>
  // JavaScript-Code
</script>
<!-- beliebiges HTML -->
<script>
  // JavaScript-Code
</script>
<!-- beliebiges HTML -->
```

Am Element („inline“, veraltet)

JavaScript lässt sich auch direkt mit einem Element verbinden.

Der folgende Code gibt z.B. eine Dialogbox aus, wenn das Element angeklickt wird:

```
<p onclick = "alert('Hallo, Welt!')">
  Ein p-Element mit JavaScript
</p>
```

Dieses Vorgehen ist allerdings unflexibel und wird eher selten benutzt , etwa zum raschen Testen (es gibt noch weitere, ebenfalls veraltete Möglichkeiten)

Notation

Jedes Statement (jede Anweisung) steht in einer eigenen Zeile

Am Ende eines Statements steht ein Semikolon ;

JS erkennt die einzelnen Statements am Semikolon, man könnte den kompletten Code also in einer einzigen Zeile schreiben – ist halt nicht mehr lesbar …

JavaScript ist case sensitive! (Groß-/Kleinschreibung wird unterschieden)

Whitespace wird ignoriert

Einzeilige Kommentare:

```
// Ein Kommentar
```

Mehrzeilige Kommentare:

```
/* Ein Kommentar
   über drei
   Zeilen          */
```

Die Browser-Werkzeuge

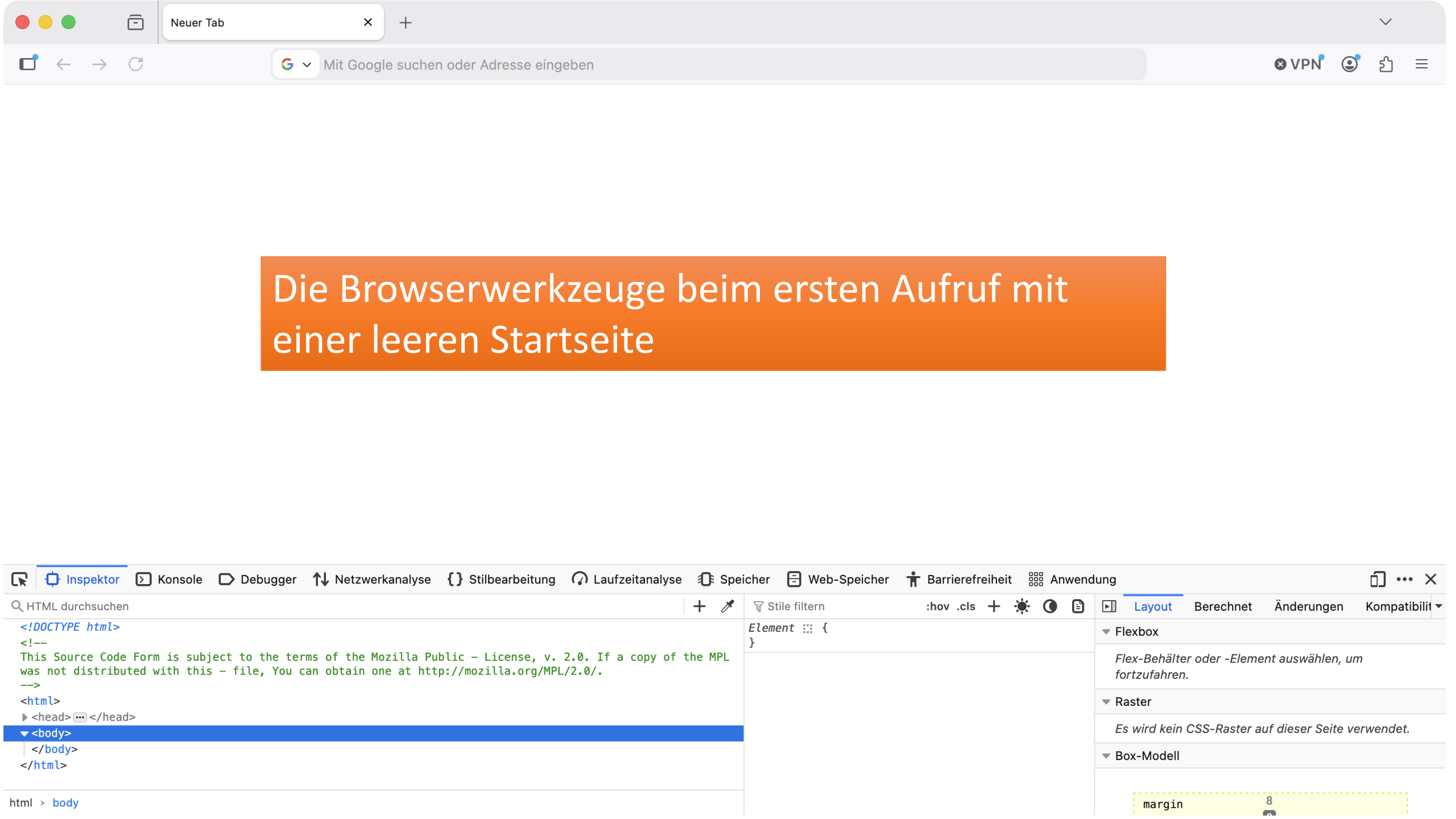
Jeder moderne Browser bietet Tools, mit denen eine Webseite detailliert analysiert werden kann

Aufgerufen werden sie in der Regel über F12 (es gibt noch andere Wege)

Zu diesen Tools gehört auch die „Konsole“. Hier lässt sich JavaScript-Code ausprobieren, ohne dass zuerst eine Webseite als Gerüst gebastelt werden muss

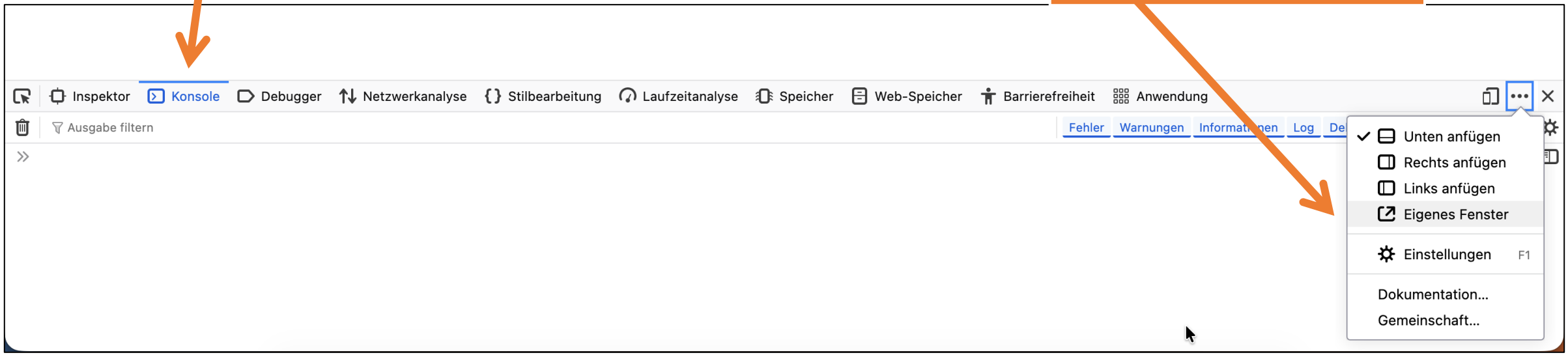
Fehler im JavaScript-Code werden nur in der Konsole angezeigt!

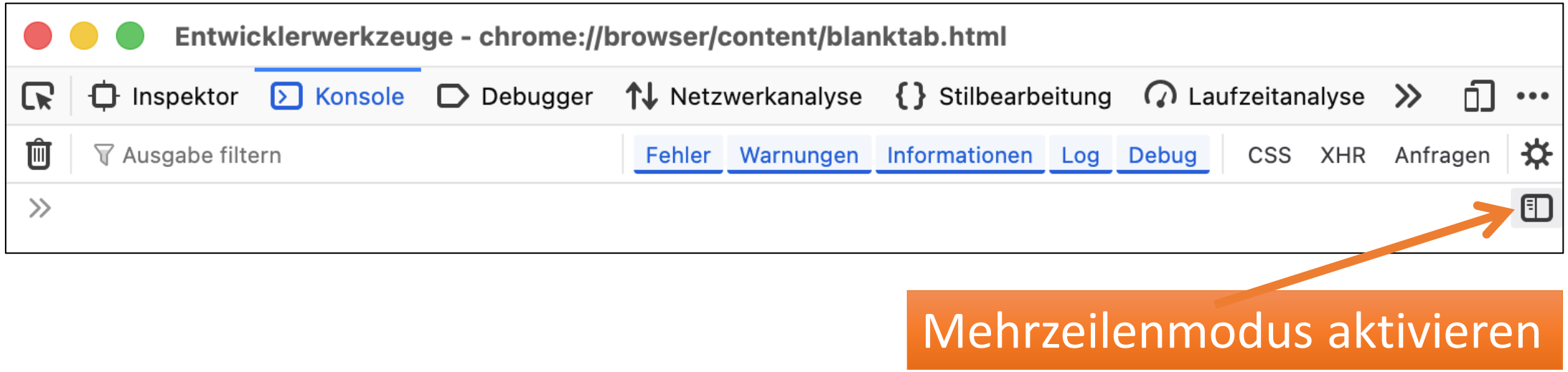
Im Kurs benutze ich gelegentlich die Konsole von Firefox

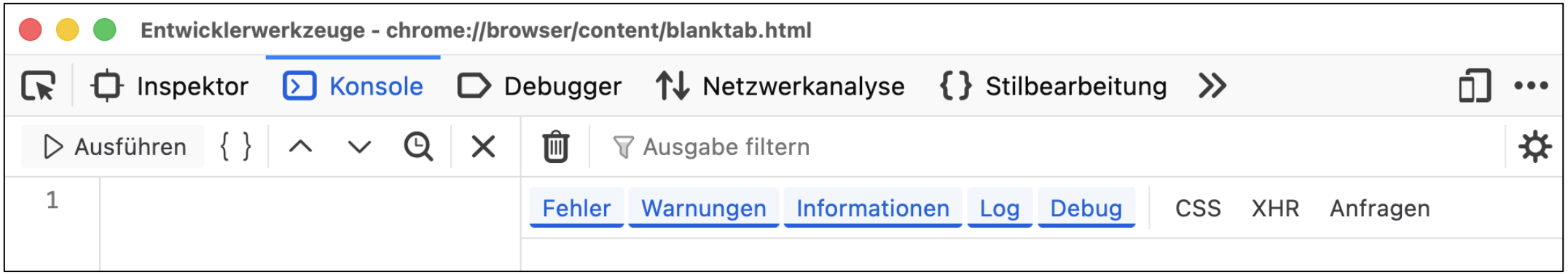

Neuer Tab
Mit Google suchen oder Adresse eingeben
VPN
Die Browserwerkzeuge beim ersten Aufruf mit einer leeren Startseite
Inspektor
Konsole
Debugger
Netzwerkanalyse
Stilbearbeitung
Laufzeitanalyse
Speicher
Web-Speicher
Barrierefreiheit
Anwendung
HTML durchsuchen
Stile filtern
:hov .cls
Layout
Berechnet
Änderungen
Kompatibilit
<!DOCTYPE html>
<!--
This Source Code Form is subject to the terms of the Mozilla Public - License, v. 2.0. If a copy of the MPL
was not distributed with this - file, You can obtain one at http://mozilla.org/MPL/2.0/.
-->
<html>
<head> </head>
<body>
</body>
</html>
Element { }
Flexbox
Flex-Behälter oder -Element auswählen, um fortzufahren.
Raster
Es wird kein CSS-Raster auf dieser Seite verwendet.
Box-Modell
margin
8
html › body

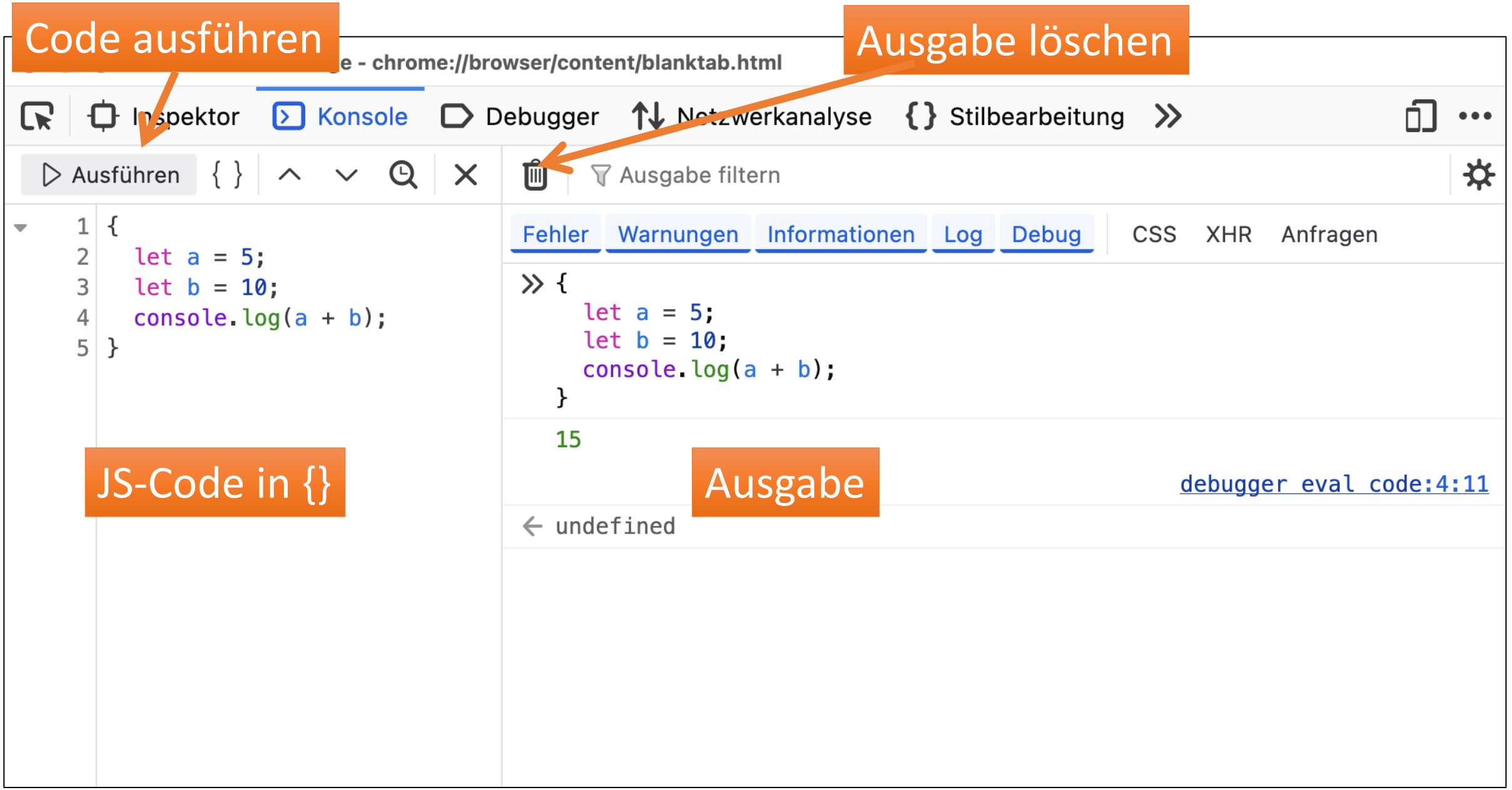
Konsole auswählen

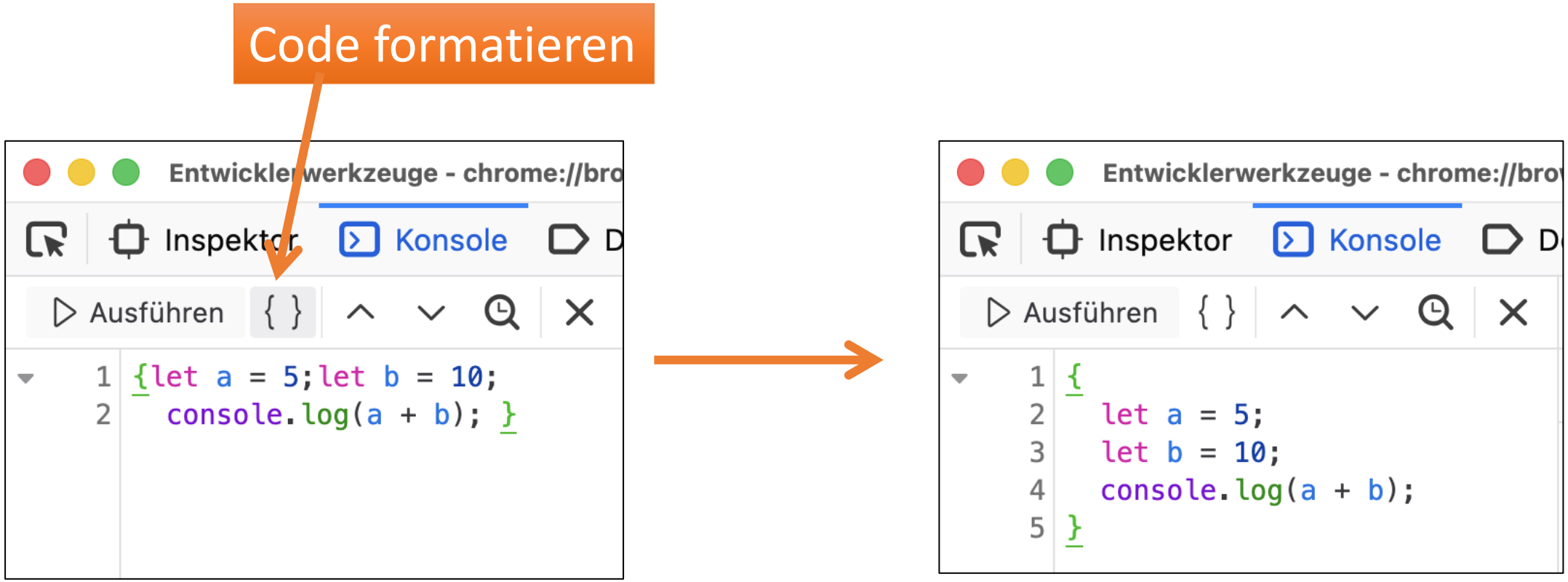
Werkzeuge in einem eigenen Fenster öffnen




Entwicklerwerkzeuge - chrome://browser/content/blanktab.html
Inspektor
Konsole
Debugger
Netzwerkanalyse
Stilbearbeitung
Laufzeitanalyse
Ausgabe filtern
Fehler
Warnungen
Informationen
Log
Debug
CSS
XHR
Anfragen
Mehrzeilenmodus aktivieren


Entwicklerwerkzeuge - chrome://browser/content/blanktab.html
Inspektor
Konsole
Debugger
Netzwerkanalyse
Stilbearbeitung
Ausführen
Ausgabe filtern
1
Fehler
Warnungen
Informationen
Log
Debug
CSS
XHR
Anfragen


Code ausführen
Ausgabe löschen
e - chrome://browser/content/blanktab.html
Inspektor
Konsole
Debugger
Netzwerkanalyse
Stilbearbeitung
Ausführen
{ }
Ausgabe filtern
Fehler
Warnungen
Informationen
Log
Debug
CSS
XHR
Anfragen
1 {
2 let a = 5;
3 let b = 10;
4 console.log(a + b);
5 }
» {
let a = 5;
let b = 10;
console.log(a + b);
}
15
debugger eval code:4:11
undefined
JS-Code in {}
Ausgabe


Code formatieren
Entwicklerwerkzeuge - chrome://bro
Inspektor
Konsole
Ausführen
{ }
1 {let a = 5;let b = 10;
2 console.log(a + b); }
Entwicklerwerkzeuge - chrome://bro
Inspektor
Konsole
Ausführen
{ }
1 {
2 let a = 5;
3 let b = 10;
4 console.log(a + b);
5 }