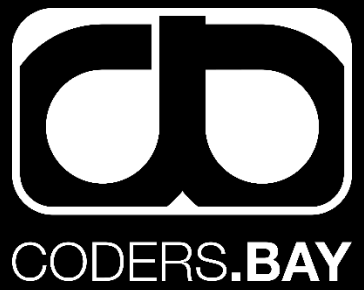


JavaScript

Jetzt muss es sich aber endlich bewegen

JavaScript

- JavaScript ist eine vollwertige Programmiersprache
  - komplexe Programme umsetzen
  - erste Version dieses Industriestandard wurde Mitte 1997 verabschiedet unter dem Namen ECMAScript
  - In schönstem Fachchinesisch eine dynamisch typisierte, prototypenbasierte, interpretierte Programmiersprache
    - Interpretiert heißt, der Programmcode wird genau in dem Moment vom Computer analysiert und in Instruktionen übersetzt, wenn er auch ausgeführt wird.

Trotz der Namensähnlichkeit hat JavaScript nichts mit Java zu tun

Das erste JavaScript

CODERS.BAY

Das **<script>**-Tag, ohne Parameter, enthält JavaScript im Tag-Body. **<script>** kann sowohl im **<head>** als auch im **<body>** der Seite stehen. Experten empfehlen, wenn möglich JavaScript am Ende des **<body>** zu laden, denn so werden sichtbare Elemente zuerst geladen, und der Betrachter bekommt schneller etwas zu sehen.

```
...
    <body>
        <script>
            alert("Hallo Welt!");
        </script>
        <noscript>
            JavaScript ist nicht aktiv!
        </noscript>
    </body>
...
```

Das ist ein Funktionsaufruf. Beim Aufruf einer Funktion können Parameter übergeben werden

Die Anführungszeichen sind wichtig, sie halten die Zeichenkette zusammen.

Jede JavaScript Anweisung wird mit einem **Semikolon** abgeschlossen.

<noscript> ist, wenn du JavaScript einbindest, fast so wichtig wie **<script>**, obwohl es im besten Fall nichts tut und auch nicht zu sehen ist. Der Inhalt von **<noscript>** wird nur angezeigt, wenn JavaScript im Browser entweder abgeschaltet ist oder der Browser zu alt ist, um JavaScript zu kennen. Letzteres kommt heute so gut wie nicht mehr vor, Ersteres dafür umso öfter. Es ist immer eine gute Idee, einen Hinweis im **<noscript>** mit in die Seite zu packen.

Externes JavaScript einbinden

Obwohl der Tag-Body leer ist, darf das **<script>**-Tag nie als leeres Tag **<script/>** geschrieben werden. Der Code wird sonst nicht ausgeführt!

```
<script src="meinskript.js"></script>
```

Absolute oder relative URLs, alles geht. Wichtig ist, dass die Skriptdatei auf **.js** endet, damit auch wirklich jeder Browser versteht, dass JavaScript drinsteckt.

Auf Seiten in HTML 4 muss das **<script>**-Tag sowohl für eingebettete als auch für externe Skripte zusätzlich das Attribut **type="text/javascript"** haben:
<script type="text/javascript" src="meinskript.js">

Zählen nach Zahlen

JavaScript kann rechnen.

```
<script>alert(7+3);</script>
```

```
alert(7+3);
document.write(7+3);
```

Das Warten auf Bestätigung ist eine Eigenschaft der **alert**-Funktion: Bis jemand den OK- Knopf drückt, wird alles auf der Seite angehalten: JavaScript, Rendering, alles.

Der Punkt in **document.write** bedeutet streng genommen, dass die Methode **write** des Objekts **document** aufgerufen wird. Du kannst es für den Augenblick wie eine einfache Funktion betrachten, die eben einen Punkt im Namen hat.

VARIABLEN

CODERS.BAY

- Variablen sind benannte Speicherstellen, eine Art Postfach, in dem wir einen Wert ablegen und ihn bei Bedarf wieder hervorholen. Es gibt also zwei Dinge zu tun: einen Wert in einer Variablen zu speichern und diesen Wert wieder hervorzuzaubern.

Eine Variablendeklaration beginnt mit dem Schlüsselwort **let**. Bevor du einer Variablen zum ersten Mal einen Wert zuweist, muss sie deklariert werden.

```
let meineVariable;

meineVariable= 3*3;

let andereVariable = 5;
```

Das Gleichheitszeichen weist der Variablen einen Wert zu. Der Variablenname muss immer links vom Gleichheitszeichen stehen, der Wert rechts. **3 = meineVariable** funktioniert nicht!

Eine Variable braucht immer einen Namen, wir müssen sie ja später auch wiederfinden können. Variablennamen dürfen mit Buchstaben oder den Zeichen _ oder **$** beginnen, weiter hinten dürfen auch Ziffern vorkommen. Variablennamen sollten immer mit einem Kleinbuchstaben beginnen, das ist zwar nicht vorgeschrieben, aber eine Konvention, an die sich fast alle JavaScript-Entwickler halten.

Deklaration und Zuweisung lassen sich in einer Zeile zusammenfassen.

Der zugewiesene Wert muss keine einfache Zahl sein, es kann auch das Ergebnis einer Berechnung zugewiesen werden. **meineVariable** hat nach dieser Zeile den Wert 9.

INFOS UND AUSGABE

CODERS.BAY

- Wenn ein Variablenname aus mehreren Wörtern besteht, ist es üblich, jedes Wort außer dem ersten mit einem Großbuchstaben zu beginnen. Diese Konvention heißt **CamelCase**
- Wenn du Beispiele aus dem Internet anschaust, wirst du oft sehen, dass Variablen mit **let** deklariert werden anstatt mit **var**. **let** ist die neuere (und schönere) Variante, wird aber noch nicht in allen verbreiteten Browsern unterstützt. Bevor ich dir den Unterschied zwischen **let** und **var** zeige, muss ich dir noch erklären, was ein Block ist.

- Eine Variable, die deklariert wurde, der aber noch kein Wert zugewiesen wurde, hat den speziellen Wert **undefined**.
- Eine Variablenzuweisung, ein Funktionsaufruf oder die Kontrollkonstrukte, die du noch kennenlernst, bezeichnet man allgemein als ein **Statement**.

```
document.write(3 * meineVariable*1);
```

Den Wert aus einer Variablen wieder auszulesen, ist so einfach, wie es nur sein kann: Der Variablenname kann überall stehen, wo auch der Wert der Variablen stehen könnte.

ZAHLENTHEORIE

- Es gibt keinen Unterschied zwischen ganzen Zahlen (den sogenannten Integer-Werten) und Kommazahlen (den Float-Werten).
- Zahlen können nicht unbegrenzt groß werden
- In JavaScript gibt es nur den Datentyp **Number**
- alle Zahlen in JavaScript als „Double precision floating point values“ (Fließkommawerte mit doppelter Genauigkeit) nach IEEE 754 behandelt werden

Auch bei JavaScript werden Nachkommastellen mit einem Punkt abgetrennt, nicht wie im deutschsprachigen Raum üblich mit einem Komma!

onClick();

- Wenn du dich an das Formularkapitel erinnerst, dann weißt du noch, dass ein Knopf mit **type="button"** von sich aus nichts tut und wir sein Verhalten mit JavaScript definieren müssen. Genau das tun wir jetzt.

*1 Der Knopf soll tun, was wir ihm sagen, und nichts anderes.

```
Schrödingers Freundin hat Schuhe. <br>
<button type="button"*1 onclick="zaehleSchuhe();"*2>Klick mich!</button>
```

*2 Das ist der mysteriöse Eventhandler. So mysteriös ist er aber gar nicht, **onclick** bedeutet schlicht, dass der JavaScript-Code aus dem Attributwert ausgeführt wird, sobald dieses Element angeklickt wird. Hier wird die Funktion **zaehleSchuhe** aufgerufen.

Wenn du **getElementById()** mit einer ID aufrufst, die es im Dokument nicht gibt, dann erhältst du einen sogenannten **null**-Wert. **null**-Wert bedeutet, dass es das gesuchte Objekt nicht gibt. Und wenn du versuchst, Methoden daran aufzurufen, endet das mit einem Fehler.
Die Zuweisung an **innerHTML** überschreibt den vorherigen Inhalt des Elements. Überhaupt ist **innerHTML** eher die Holzhammermethode, um den Seiteninhalt zu manipulieren.

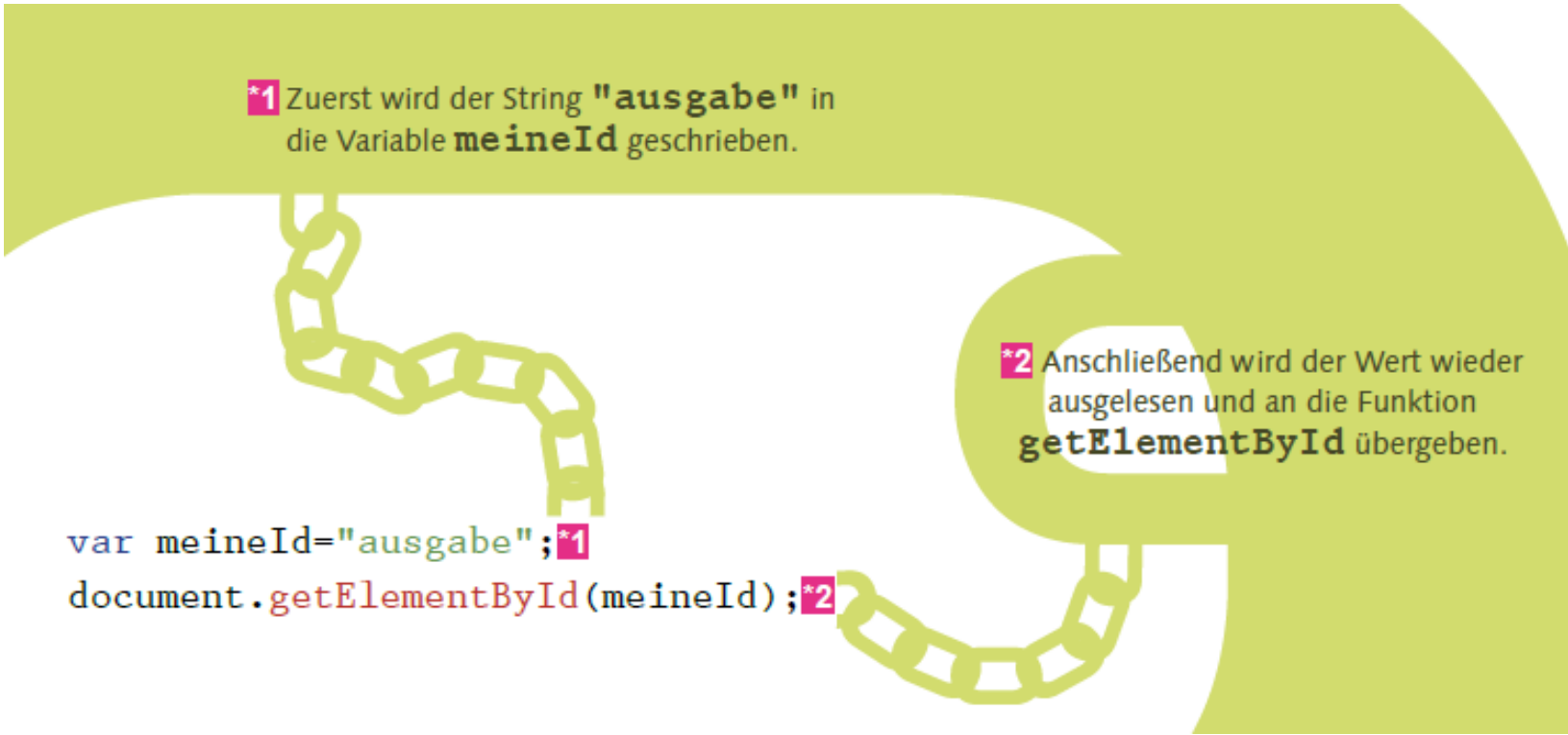
Woher weiß ich, wenn ein Fehler auftritt?

- Um Fehler zu sehen zu bekommen, muss man im Browser die Fehlerkonsole öffnen, dort wird alles angezeigt, was schiefgeht. Natürlich ist die mal wieder in jedem Browser an einer anderen Stelle versteckt. Im Internet Explorer findet man sie unter „Einstellungen“ • „Entwicklertools“ • „Konsole“, in Firefox unter „Web-Entwickler“ • „Fehlerkonsole“, bei Chrome unter „Tools“ • „JavaScript-Konsole“ und schließlich bei Safari unter „Entwickler“ • „Webinformationen einblenden“ • „Konsole“. Aber zumindest sind sich alle Browser einig, dass ein Methodenaufruf an **null** ein Fehler ist.

Zeichen, Zeichen, Zeichenkette

CODERS.BAY

- die Zeichenkette - den String
- Immer, wenn in JavaScript etwas in Anführungszeichen steht, handelt es sich um einen String
- Wenn ein String zwischen Anführungszeichen im Code steht, nennt man das ein Stringliteral. Wir werden gleich sehen, dass das nicht die einzige Möglichkeit ist, an einen neuen String zu gelangen.
- Mit Strings kann man mit dem Operator **+**, zwei Strings verketten



Zeichen, Zeichen, Zeichenkette

```
var name = "Schrödinger";
var gruss = "Hallo " +*1 name +*1 ", wie geht es dir heute?"*2
```

*1 Keine Addition, sondern **Stringkonkatenation**: Es wird ein neuer String erzeugt, indem der String links vom + und der String rechts vom + hintereinandergeschrieben werden. Genau wie Addition bei Zahlen kann man das auch mehrmals nacheinander machen.

*2 Der Variablen `gruss` wird ein neuer String zugewiesen. Sein Inhalt ist „Hallo Schrödinger, wie geht es dir heute?". Genau wie beim Rechnen mit Zahlen gibt es keine Verbindung mehr zur Variable `name`, ihr Wert kann sich ändern, ohne dass `gruss` davon betroffen ist.

[Achtung]

Bei der Stringkonkatenation musst du selbst auf Leerzeichen achten, JavaScript fügt keine ein.

Zeichen, Zeichen, Zeichenkette

CODERS.BAY

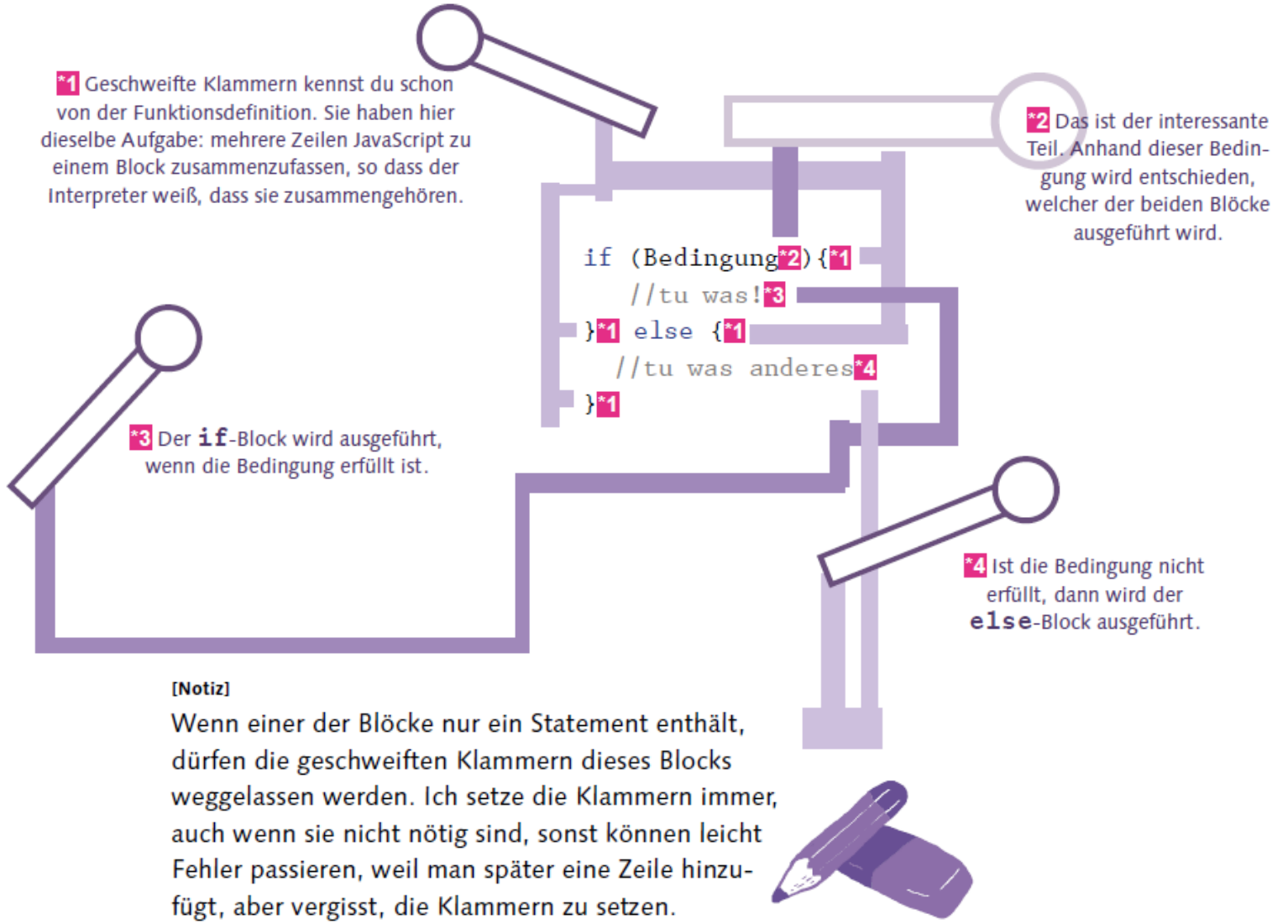
- Strings sind Objekte sind, die eigene Methoden und Eigenschaften haben. Eine Eigenschaft, ist **length**. Darin steht, wie viele Zeichen der String enthält.
- Methode **charAt()** findet das Zeichen an der n-ten Stelle des Strings
- Bei allen Funktionen, bei denen mit einer Position im String gearbeitet wird, hat das erste Zeichen den Index 0.
- Möchte man wissen, an welcher Stelle im String ein bestimmtes Zeichen vorkommt, helfen dabei die Funktion **indexOf()** und **lastIndexOf()**. Beide nehmen als ersten Parameter einen String und suchen dann, wo dieser String in „ihrem“ String vorkommt. Sie geben den Index zurück, an dem sie etwas gefunden haben. Dabei fängt **indexOf()** vorne im String an zu suchen, **lastIndexOf()** sucht von hinten

Wenn … dann …

- Der Unterschied zwischen **==** und **===** (bzw. **!=** und **!==**) liegt darin, dass **==** und **!=** Typen automatisch umwandeln, so wie du es auch schon bei **+** mit Zahlen und Strings gesehen hast. Deshalb ist **2 == "2"** wahr, **2 === "2"** aber falsch.

- Beim Vergleich, ob zwei Zahlen gleich sind, müssen es zwei Gleichheitszeichen sein! Ein einzelnes ist für die Zuweisung an eine Variable, zum Vergleich sind es zwei.

>	größer als
>=	größer oder gleich
==	gleich
<=	kleiner oder gleich
<	kleiner
!=	ungleich
===	gleich ohne Typkonvertierung
!==	ungleich ohne Typkonvertierung



[Notiz]

Wenn einer der Blöcke nur ein Statement enthält, dürfen die geschweiften Klammern dieses Blocks weggelassen werden. Ich setze die Klammern immer, auch wenn sie nicht nötig sind, sonst können leicht Fehler passieren, weil man später eine Zeile hinzufügt, aber vergisst, die Klammern zu setzen.

AUFGABE

CODERS.BAY

- Lese eine Zeichenkette ein und überprüfe
  - ob das Wort „ein“ darin enthalten ist
  - Wenn ja – wie oft ist es enthalten?
  - An welcher Stelle tritt es auf?

  - Hilfestellung:
    - **charAt()**

  - Eingabe:
    - Drei kleine Jägermeister sind ein schönes Gespann.
  - Ausgabe:
    - Ja
    - 2
    - 8 und 31

AUFGABE

CODERS.BAY

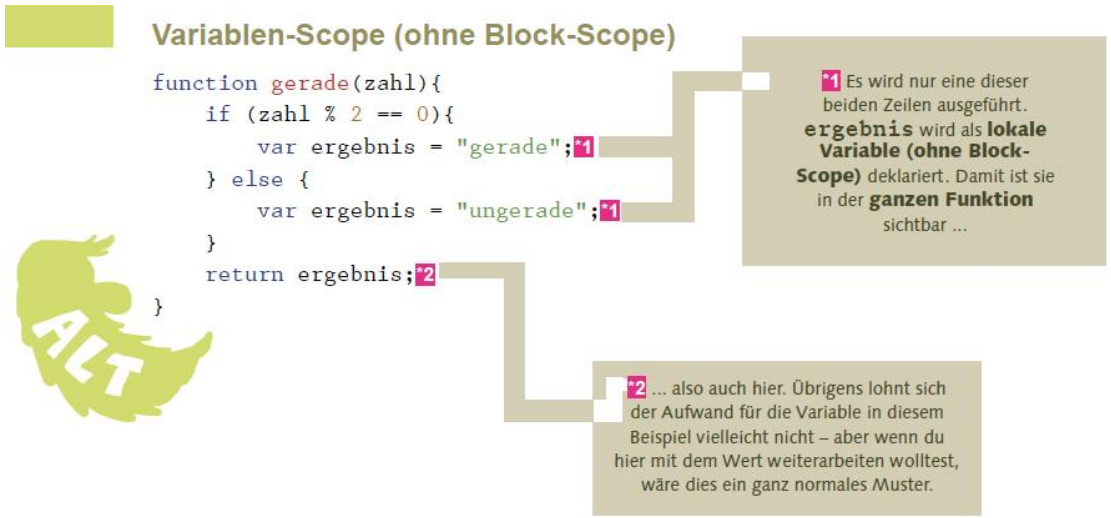
- Lese eine 2 Zeichenketten ein
  - Die erste Zeichenkette ist ein Text
  - Die zweite Zeichenkette ist der String auf den überprüft wird
  - Vor und nach dem String der überprüft wird soll in der Ausgabe ein * stehen
  - Gib den Text danach im html aus

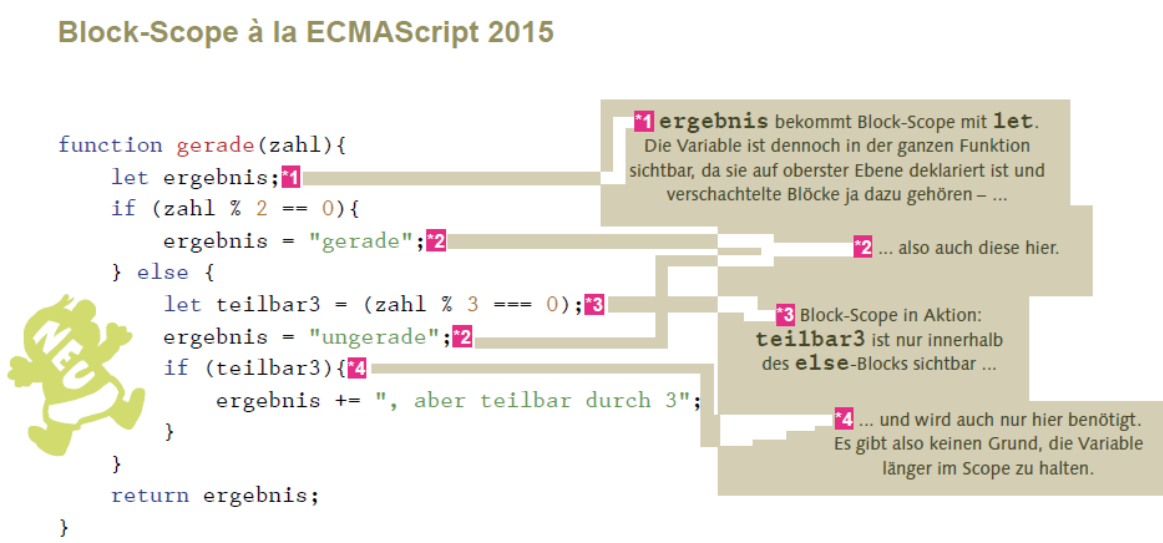
  - Hilfestellung:
    - **Document.write();**

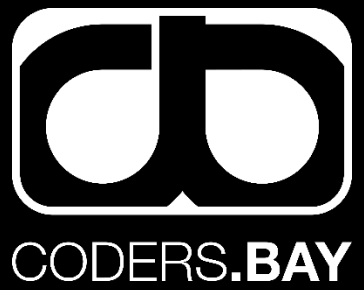
  - Eingabe:
    - Drei kleine Jägermeister sind ein schönes Gespann.
    - e
  - Ausgabe:
    - Dr*e*i kl*e*in*e* Jäg*e*rm*e*ist*e*r sind *e*in schön*e*s G*e*spann.

Block-Scope

- *Globale Variablen* werden außerhalb einer Funktion deklariert und sind überall sichtbar.
- *Lokale Variablen* werden innerhalb einer Funktion deklariert und sind nur in dieser Funktion sichtbar – dort aber überall, die Codeblöcke spielen keine Rolle!
- Mit dem Schlüsselwort **let** gibt es einen dritten:
- Variablen mit Block-Scope sind nur in dem Block sichtbar, in dem sie deklariert wurden. Inklusive darin verschachtelter Blöcke. Damit eine Variable Block-Scope bekommt, deklariere sie mit **let** statt mit **var**.
- Block-Scope funktioniert in IE ab Version 11, in Edge, Firefox und Chrome.







Ende