Softwareentwicklung - JavaScript

Inhalt

[Was ist JavaScript? 2](#_Toc113396161)

[Was kann man damit machen? 3](#_Toc113396162)

[Wo läuft JavaScript? 3](#_Toc113396163)

[JavaScript vs ECMAScript? 3](#_Toc113396164)

[Demo 1 4](#_Toc113396165)

[Einrichten der Entwicklungsumgebung 5](#_Toc113396166)

[Der erste JavaScript Code 6](#_Toc113396167)

[Separation of Concerns 7](#_Toc113396168)

[Variablen 9](#_Toc113396169)

[Regeln für Variablenbezeichnungen 11](#_Toc113396170)

[Konstanten 11](#_Toc113396171)

[Datentypen für Variablen 12](#_Toc113396172)

[Dynamische Typisierung 13](#_Toc113396173)

[Objects 15](#_Toc113396174)

[Arrays 18](#_Toc113396175)

[Functions 21](#_Toc113396176)

# Was ist JavaScript?

JavaScript ist einer der populärsten Programmiersprache der Welt und wächst an Popularität schneller als irgendeine andere Programmiersprache.

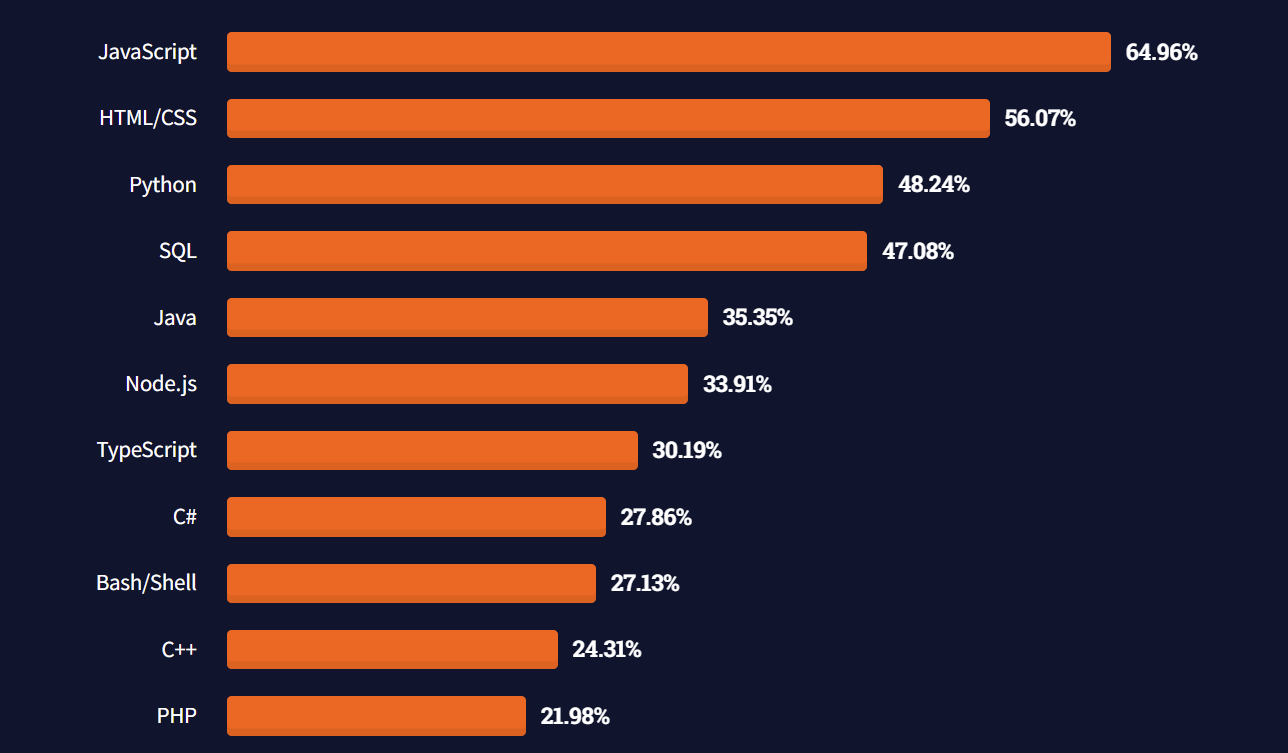


Abbildung 1: Aus Stack Overflow Developer Survey 2021

Viele bekannte Firmen erstellen ganze Applikationen damit.

Mittels JavaScript kann das ganze Spektrum der Softwareentwicklung abgedeckt werden. Es kann eingesetzt werden für:

* Frontend Entwicklung
* Backend Entwicklung
* Fullstack Entwicklung

# Was kann man damit machen?

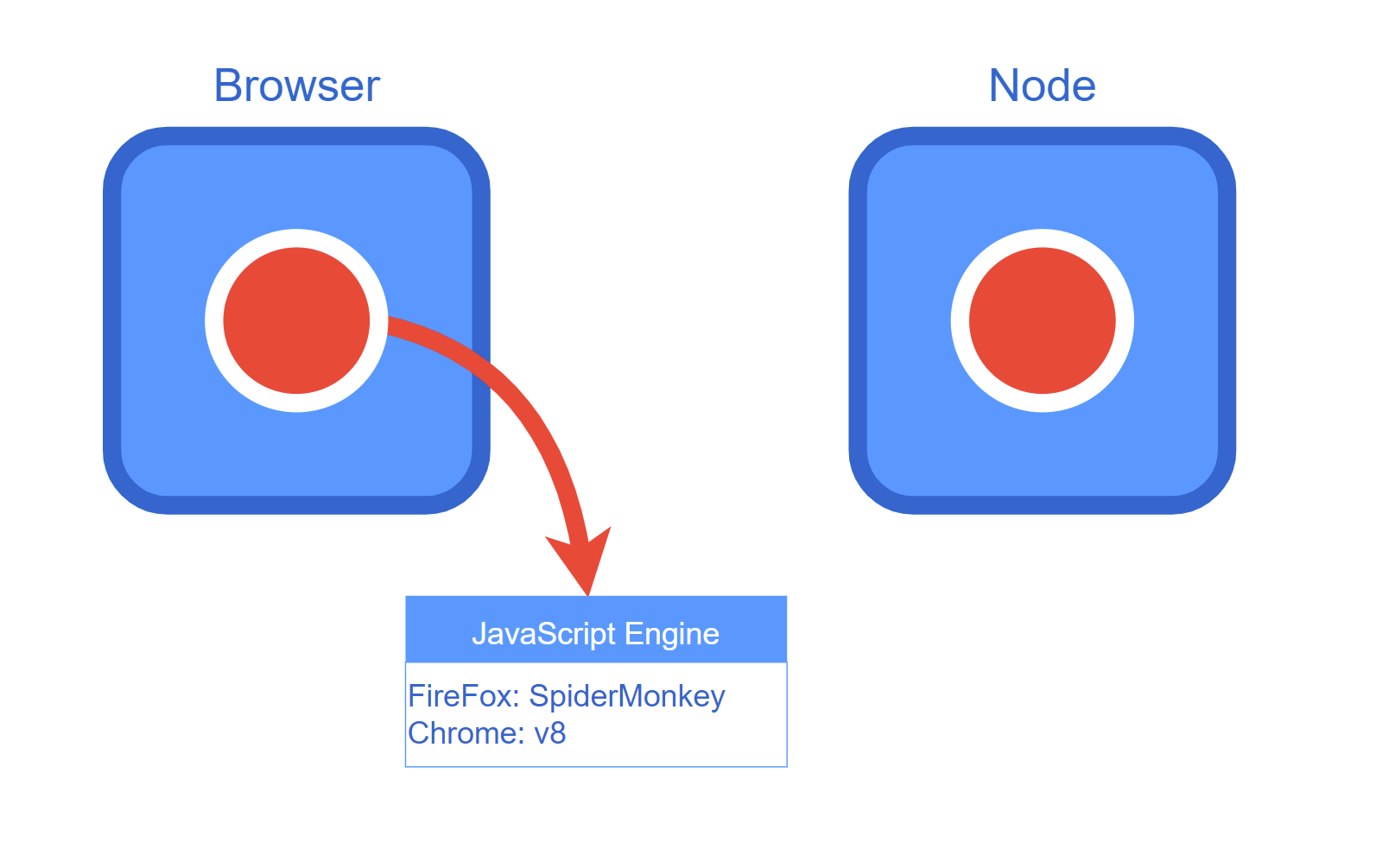
JavaScript wurde für den Browser entwickelt, um in Webseiten Logiken einbauen zu können. Durch die Weiterentwicklung der Sprache durch die OpenSource-Community und vielen großen Firmen wie Facebook, Google, … kann man mit JavaScript mittlerweile eine Vielzahl an Anwendungen entwickeln:

* Web / Mobile Apps
* Realtime Network Apps
* Commandline Tools
* Games
* …

# Wo läuft JavaScript?

Ursprünglich wurde JavaScript entworfen, um in einem Browser zu laufen. Damit der Browser mit dem JavaScript Code etwas anfangen kann (d.h. den Code interpretieren und ausführen) benötigt er eine JavaScript Engine.

2008 hat ein OpenSource-Entwickler die JavaScript Engine aus Chrome, v8, in ein C++-Programm implementiert. Dadurch ist es möglich den JavaScript Code auch außerhalb des Browsers laufen zu lassen um damit Server Anwendung zu Entwickeln. Dieses C++-Programm heißt Node.



v8

Abbildung 2: Wo läuft JavaScript

# JavaScript vs ECMAScript?

ECMAScript ist die Norm (vgl. VDI, DIN, ISO) für die Sprache JavaScript. JavaScript ist die Implementierung der Norm. Aktuell ist ECMAScript 2021.

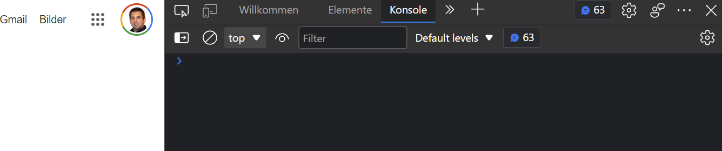
# Demo 1

JavaScript kann ganz einfach im Browser ausprobieren:

In einer geöffneten Website (z.B.: [www.google.at](http://www.google.at)) auf die rechte Maustaste klicken und im Menü „Untersuchen“ auswählen oder die Tastenkombination „Strg + Shift + i“ drücken. Das öffnet die „Developer Tools“. In diesem Fenster auf „Console“ wechseln.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung



In der „Konsole“ kann man nun jeden erdenklichen, gültigen JavaScript Code eingeben und ausprobieren:

* console.log(„Hello World“);
* 2 + 2
* alert(„yo“);

# Einrichten der Entwicklungsumgebung

Um sich das Erstellen von JavaScript Programmen einfacher zu machen, ist es sinnvoll dies in einem Code-Editor zu machen. Es gibt eine Vielzahl davon. Ein kleiner Auszug

* Visual Studio Code (VSCode)
* Sublime Text
* Atom
* Notepad++
* …

VSCode ist zu empfehlen, da es dazu eine Vielzahl von hilfreichen Plugins gibt, die einem die Arbeit erleichtert.

Neben einem Code Editor, empfiehlt sich auch Node zu installieren, da viele Frameworks und Tools für das Frontend und da Backend via Node installiert werden (NPM).

Für die weiteren Beispiele benötigen wir einen Ordner, in dem wir die Dateien ablegen. Diesen Ordner öffnen wir in VSCode und erstellen uns eine Datei mit dem Namen „index.html“ und erstellen uns ein HTML Gerüst mit „!+Tab“. Dieses HTML Gerüst dient als Host für unseren JavaScript Code.

Am einfachste arbeitet man mit dem LiveServer, eine Erweiterung für VSCode. Um zu überprüfen, dass alles richtig eingerichtet ist, sollte man die Seite in LiveServer starten. Im „Body“ Bereich der „index.hml“ wird ein „<h1>Hello World</h1>“ eingefügt. Nach dem Speichern sollte der Text, in dem von LiveServer gestarteten Browser angezeigt werden.



Abbildung 3: index.html mit LiveServer im Browser

# Der erste JavaScript Code

In einem HTML Dokument gibt es 2 Möglichkeiten JavaScript Code einzubauen. Im „<head>“ Bereich und am Ende des „body“ Bereichs. Zweiteres ist zu empfehlen, da der Browser, wenn er das HTML Dokument abarbeitet, dabei von oben nach unten arbeitet. Dadurch wird der Code erst ausgeführt, wenn die Seite geladen wurde. Wenn der Code im „<head>“ Bereich geladen wird, wird die Seite erst übersetzt, wenn der gesamte JavaScript Code ausgeführt wurde. Das führt zu einem schlechten Verhalten für den Benutzer. Dieser bekommt die eigentliche Seite erst zu sehen, wenn der code ausgeführt wurde. Weiters wird im Code oft auf Elemente der Webseite zugegriffen. Diese sind aber erst verfügbar, wenn die Seite geladen wurde. Das ist nicht der Fall, wenn das Script im Kopf-Bereich steht.

Am Ende des „<body>“ Bereichs wird nun ein „<script></script>“ Element eingefügt. Der Code der zwischen diesen beiden „Tags“ steht wird ausgeführt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 4: Der erste JavaScript Code

„console.log“ ist ein Statement. Innerhalb der Klammern „‘Hello World‘“ ist ein String. Ein String ist eine Reihe von Zeichen. Über dem Statement steht Text nach 2 „//“. Dieser Text nennt sich Kommentar und dient den Code zu Beschreiben.

Nach dem Speichern wird in der Konsole der Developer Tools des LiveScript Browsers „Hello World“ geschrieben.

Gratulation, das ist der erste JavaScript Code.

# Separation of Concerns

In der Informatik ist “Separation of Concerns“ ein Entwurfsmuster, um die Aufgaben des Programms in Verantwortungsbereiche zu unterteilen.

Ein Beispiel:

In einem Haus gibt es mehrere Räume. Das Gewand ist im Kasten im Schlafzimmer, Die Speisen sind im Kühlschrank in der Küche und das Geschirr ist in einem Schrank im Esszimmer. Niemand würde das Essen im Schlafzimmer lagern oder das Gewand im Esszimmer. Alles hat seinen Platz.

Dies kann auch am Beispiel einer Webseite erklärt werden. In einer Webseite sollen Daten (Text, Bilder, …) in einer Struktur aufgebaut werden. Damit das auch für den User ansprechend ist, muss diese Struktur noch dementsprechend präsentiert werden (Farben, Schriftgrößen, …). Um dem User zu ermögliche mit der Seite zu interagieren, ist eine Form von Programlogik erforderlich.

Daher werden alle Webseiten nachfolgendem Prinzip aufgebaut:

* HTML ist für den strukturierten Aufbau zuständig
* CSS für Darstellung
* und JavaScript für die Interaktion

Damit diese Aufgeben Gebiete getrennt werden können, werden sie in eigene Dateien abgelegt. HTML in einer Datei mit der Endung .html, CSS in einer Datei mit der Endung .css und JavaScript in einer Datei mit der Endung .js. Natürlich kann es von jeder Datei mehrere geben.

Wie verwendet man nun eine JavaScript Datei in seiner HTML Seite? Das „<script>“ Tag hat mehrere Attribute. Eines davon ist das „src“ Attribut. Darin kann man den Pfad zu einer JavaScript Datei angeben.

In VSCode soll eine neue Datei mit dem Namen „index.js“ erstellt werden. In die Datei wird der Inhalt des „<script>“ Tags kopiert. Als nächstes kann man alles zwischen den „<script>“ Tags löschen und dem „<script>“ Tag ein Attribut „src“ mit dem Pfad zu der JavaScript Datei „index.js“ hinzugefügt werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

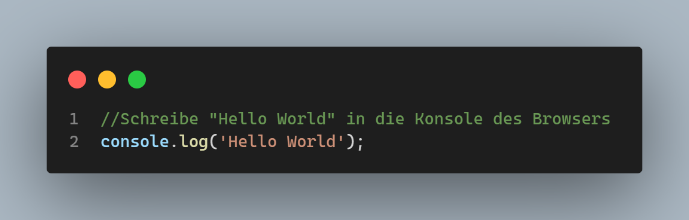


Abbildung 5: index.js und index.html

Im LiveServer hat sich das Ergebnis nicht verändert. Jedoch ist der Aufbau von der Programm Logik getrennt.

Den JavaScript code kann man nun auch mit Node ausführen, also komplett ohne Browser.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 6: index.js in Node

# Variablen

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Abbildung 7: Variable im Speicher

Variablen werden verwendet, um Informationen temporär im Speicher des Computers abzulegen. Diesem Speicherort wird ein Name zugewiesen, der Variablenname, um die Information auslesen zu können.

Variablen sind vergleichbar mit Schachteln oder Boxen, mit denen man seine Sache organisiert. Damit man weiß was enthalten ist, beschriftet man die Schachteln. So finden man den Inhalt später leichter.



Abbildung 8: Schachteln mit Beschriftung

Eine Variable ist wie so eine Schachtel. In die Schachtel legen wir den Wert, den wir uns merken möchten, die Daten. Die Schachtel beschriften wir mit einem Text, den Variablennamen.

Ein Beispiel:

Bevor ES6, Variablen wurden mit dem „var“-Schlüsselwort deklariert. Mit ES6 kann zwar die Variable immer noch mit „var“ deklariert werden, aber dies hat Nachteile, diese werden wir später noch näher betrachten. In der neuen Version werden Variablen mit „let“ deklariert.

D.h. die Variablendeklaration wird mit „let“ eingeleitet, dann folgt der Name der Variablen und danach muss die Deklaration mittels „;“ terminiert werden.

Nun geben wir die Variable in der Konsole mittels „console.log(name)“ aus. Auch dieses Statement müssen wir mit einem „;“ terminieren.



Abbildung 9: Beispiel Variablendeklaration

In der Konsole wir „undefined“ ausgeben. Das Bedeutet, dass Variablen bei der Deklaration standardmäßig den Wert „undefined“ haben. Man kann aber die Variable optional bei der Deklaration initialisieren. Nach der Deklaration schreibt man ein „=“ und dann z.B. einen String, also eine reihe von Zeichen zwischen Anführungszeichen. Man kann einfache als auch doppelte Anführungszeichen verwenden.



Abbildung 10: Variablendeklaration mit Initialisierung

Nun ist der Output auf der Konsole nicht mehr „undefined“ sondern der Text, in unserm Fall „Michael“, den wir in der Variable gespeichert haben.

## Regeln für Variablenbezeichnungen

1. Es darf kein reserviertes Schlüsselwort sein z.B.: let, if, else, const, …
2. Es sollte eine aussagekräftige Bezeichnung sein (Also nicht a, a1, a2, …)
3. Sie dürfen nicht mit einer Nummer beginnen (1name)
4. Sie dürfen keine Leerzeichen oder Bindestriche enthalten (-). Wenn man einen Variablennamen aus mehreren Wörtern bilden möchte, so schreibt man die Wörter aneinander. In JS hat sich die „Camel-Notiation“ etabliert. Das bedeutet, dass das erste Wort mit einem Kleinbuchstaben beginnt und jedes weitere Wort dann mit einem Großbuchstaben: „firstName“.
5. Variablennamen sind Case-sensitive

## Konstanten

Die Werte der Variablen können an jeder Stelle im Code verändert werden. Es gibt Situationen, in denen man aber nicht möchte, dass sich der Wert einer Variablen ändert. In diesem Fall verwendet man keine Variable, sondern eine Konstante. Sollte im Code versucht werden, den Wert einer Konstatne zu ändern, kommt es zu einem Fehler.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 11: Variable vs Konstante mit Fehler

Im code sollte man immer, wenn man den Wert nicht verändern muss, „const“ statt „let“ verwenden.

## Datentypen für Variablen

In JS gibt es 2 Arten von Datentypen:

* Primitives/Value Types (einfache Datentypen)
* Reference Types (Objekte)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 12: Datentypen in Javascript

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 13: Datentypen mit Beispielen

Wie man sieht, gibt es in JS nur einen Typ für Zahl. Also sowohl eine Zahl mit als auch eine Zahl ohne Kommastelle ist immer vom Typ „number“. In anderen Sprachen gibt es für Kommazahlen eigene Datentypen.

## Dynamische Typisierung

Etwas was JS von vielen anderen Programmiersprachen unterscheidet ist, dass JS eine dynamische Sprache ist. Es gibt zwei Arten von Programmiersprachen

* Statische (Statically-typed)
* Dynamische (Dynamically-typed)

In statischen Sprachen muss bei der Deklaration von Variablen der Typ angegeben werden:



Abbildung 14: Statische Typdefinition am Beispiel C#

der Typ kann später nicht mehr geändert werden.

In einer dynamischen Sprache kann der Typ einer Variablen während der Ausführungszeit geändert werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 15: Dynamische Typisierung

Weiter Typen aus dem Beispiel:

Ein Bild, das Text, schwarz, Bildschirm, Screenshot enthält.

Automatisch generierte Beschreibung

Interessant ist hier, dass der Wert von „lastName“ und der Typ von „lastName“ jeweils „undefined“ ist. Ist aber auch klar, da „undefined“ ja ein eigener Typ ist und ein Value Type also muss man ihm einen Wert zuweisen können.

Warum ist null ein „object“ und was ist ein Object? Das ist Thema des nächsten Punkts…

## Objects

Objekte gehören, so wie bereits vorhergesehen, zu den „Reference types“. Ein Objekt in JS, aber auch in allen anderen Programmiersprachen, ist wie ein Objekt im echten Leben. Man stelle sich eine Person vor. Diese Person hat einen Vornamen, einen Nachnamen, eine Adresse, ein Alter, hat eine gewisse Größe und Gewicht, usw. Das sind alles Eigenschaften einer Person. Dieses Konzept existiert auch in JS.

Um die Eigenschaften einer Person abzubilden, könnte ich Variablen anlegen, die diese Eigenschaften speichern:

Ein Bild, das Text, Uhr enthält.

Automatisch generierte Beschreibung

Abbildung 16: Eigenschaften als Variablen

Das würde aber bald unübersichtlich werden. Man stelle sich vor wir wollten die Eigenschaften mehrere Personen verarbeiten (name\_Person1, name\_Person2, usw.). Viel Saubere ist es, diese Eigenschaften in einem Objekt Person abzulegen. In JS werden Objekte ähnlich wie variablen deklariert. Lediglich wird als Wert für die Variable das Objekt-Literal „{}“ verwendet. Innerhalb dieser geschwungenen Klammern werden eine oder mehrere Eigenschaften als Schlüssel-Wert-Paare geschrieben. Die Wert-Schlüssel-Paare werden mit „Kommas“ getrennt. Ein Schlüssel-Wert-Paar besteht aus dem Key der mittels Doppelpunkts vom Wert getrennt ist.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 17: Ein JS Objekt mit zwei Eigenschaften

In der Abbildung wurde ein Objekt „person“ mit zwei Eigenschaften (Schlüssel-Wert-Paaren) erstellt. Wenn man nun mittels console.log(person) das Objekt in der Konsole ausgibt, erhält man in der Konsole das „person“-Objekt.

Ein Bild, das Text, Gerät, Anzeige enthält.

Automatisch generierte Beschreibung

Abbildung 18: Konsolenausgabe des Objekts "person"

Man sieht in der Ausgabe die Eigenschaften des Objekts die innerhalb der Objekt-Literale. Nachdem wir nun ein Objekt haben, dass die Daten speichert, möchte man auch damit arbeiten um z.B.: den Namen der Person zu ändern. Es gibt zwei Möglichkeiten, um auf die Eigenschaften eines Objekts zuzugreifen. Die erste ist die „Dot Notation“:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 19: Zugriff auf Eigenschaften eines Objekts mittels "Dot Notation"

Mittels Variable und „.<Eigenschaft>“ kann man die Eigenschaften eines Objekts verändern und auch auf diese Zugreifen (z.B.: in die Konsole ausgeben)

Eine weitere Möglichkeit, um mit den Eigenschaften zu arbeiten ist die „Bracket Notation“:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 20: Zugriff auf Eigenschaften eines Objekts mittels "Bracket Notation"

Mit zwei eckigen Klammern und den Namen der Eigenschaft hinter der Objektvariable kann man die Eigenschaften auch verändern und abfragen.

Welche Variante ist die Bessere? Meist wird die „Dot Notation“ verwendet, da es lesbarer ist. Mittels „Bracket Notation“ ist es aber möglich auf die Eigenschaft zuzugreifen, auch wenn man den Namen der Eigenschaft erst zur Laufzeit kennt. Wenn man z.B.: eine Anwendung hat, in der der User die gewünschte Eigenschaft, die er anzeigen möchte, auswählen kann. Dann wissen wir nicht beim Schreiben der Anwendung welche Eigenschaft später gewählt wird. Wenn man die Auswahl wiederum in eine Variable schreibt, kann man mittels der Variable auch auf die Eigenschaft zugreifen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 21: Zugriff auf Eigenschaften eines Objekts via Varible

In die Konsole wird hier „Maria“ geschrieben, da dies der Wert der Eigenschaft „name“ ist, die dynamisch über die Variable „selection“ ermittelt wird.

## Arrays

Gelegentlich muss man in seinen Anwendungen nicht nur mit einem Objekt oder eine Variable arbeiten, sondern mit einer List von Elementen. Etwa ein Einkaufswagen in einem Webshop mit einer Liste von getätigten Einkäufen oder mit einer Liste von Farben aus denen der User wählen kann. In JS speichert man diese List von Daten in einem Array. Ein Array wird ähnlich wie ein Objekt mit einem Literal initialisiert. In diesem Fall mit einem „Array Literal“, zwei Eckige Klammern „[]“. Wenn die Variable nur mit leeren eckigen Klammern initialisiert wird, dann ist es ein leeres Array. Wenn man Werte zwischen die Klammern schreibt, dann wird das Array mit diesen Werten initialisiert.



Abbildung 22: Initialisierung eines leeren Arrays und eines mit Farben

Gibt man das Array „selectedColors” mit “console.log” auf die Konsole aus, sieht man das Array mit den „Array Literal“ und den beiden Werten.

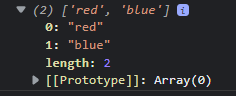


Abbildung 23: Konsolenausgabe des Arrays

Mann erkennt auch, dass jeder Eintrag im Array einen Index hat. In JS beginnt ein Array Index mit 0. (Hier 0 bei „red“). Mit diesem Index kann man auch auf die Elemente eines Arrays zugreifen. Möchte ich das erste Element dann mit 0, das zweite mit 1, usw.

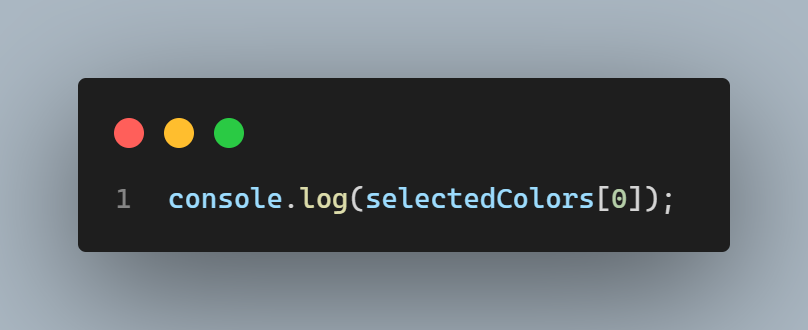


Abbildung 24: Zugriff auf das erste Element im Array

Im Konsolenfenster sehen wir nun „red“, wie erwartet.

JS ist eine dynamische Sprache. So wie man den Typen einer Variable ändern kann, kann auch der Inhalt eines Array geändert werden. Es können Elemente hinzu und entfernt werden. Es können sogar verschieden Elemente (z.B.: Text und Zahl) in ein Array geschrieben werden. Dies kann über den Index erfolgen. Im Snippet fügen wir an der dritten Stelle (Index 2) eine neue Farbe mit dem Wert „green“ ein.

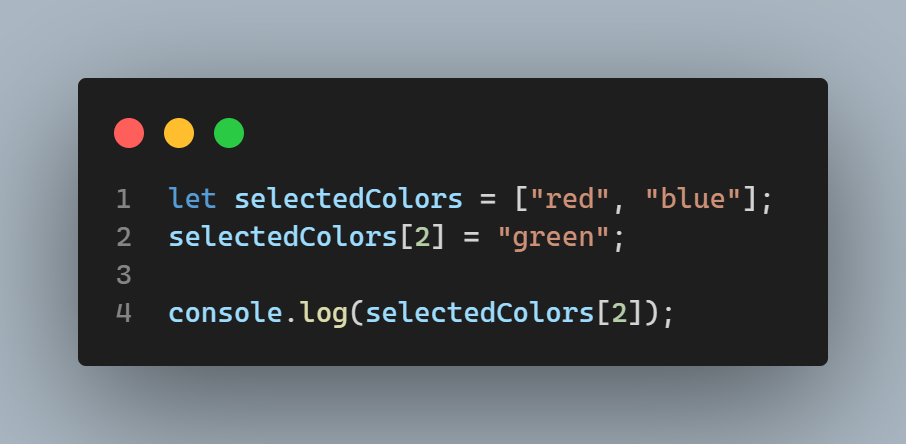


Abbildung 25: Hinzufügen eines neuen Elements zum Array

Da verschiedene Typen von Werten einem Array hinzugefügt werden können, kann man auch eine Zahl an der vierten Stellen einfügen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 26: Verschieden Typen von Werten können einem Array hinzugefügt werden

In der Konsole wird nun 34 geschrieben. Wenn man das gesamte Array ausgibt, so beinhaltet es 3 Strings und eine Zahl.



Abbildung 27: Ausgabe der verschiedenen Werte des Arrays

Technisch gesehen sind Arrays in JS Objekte. Man kann das mittels „typeof selectedColors“ überprüfen. Die Ausgabe ist „object“.

Da Arrays Objekte sind, hat es auch Eigenschaften. Diese Eigenschaften können mittels „Dot Notation“ verwendet werden.

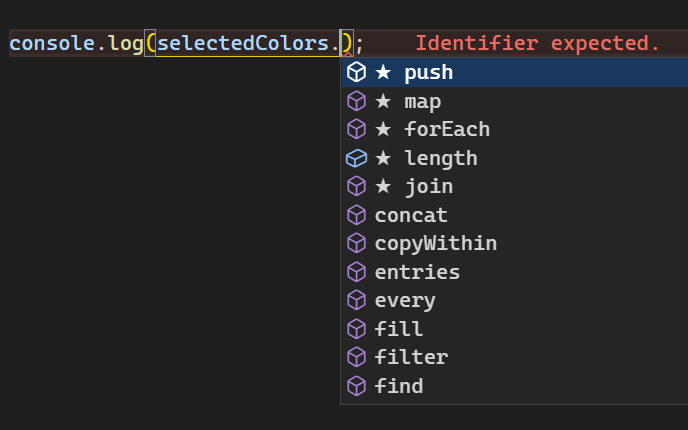


Abbildung 28: Ausschnitt von Eigenschaften eines Arrays

Jedes Mal wenn man ein Array deklariert, erhält es automatisch diese Eigenschaften. Diese mussten von uns nicht definiert werden. Sie wurden durch „Magie“ dem Array hinzugefügt. Diese Magie nennt sich Vererbung, da ein Array von einem Prototyp erbt, bei dem all diese Eigenschaften definiert wurden. Prototyps in JS werden hier aber nicht behandelt.

Eine oft verwendete Eigenschaft ist die „length“ Eigenschaft. Sie gibt an wieviel Elemente sich im Array befinden.

In unserm Fall gibt ein „console.log(selectedColors.length)” den Wert 4 auf der Konsole aus.

Dies soll als Überblick über Arrays genügen. Es gäbe noch seitenweise über die verschiedenen Eigenschaften und Methoden des Array-Objekts zu schreiben. Wichtig ist für jetzt, ein Array ist ein Objekt in dem mehre Elemente abgelegt werden können.

## Functions

Ein weitere „Reference Type“ neben Objekte und Arrays sind die Funktionen. Die Funktionen sind fundamentale Bausteine von JavaScript. Sie kapseln ein Set an Befehlen oder berechnet Werte und geben diese zurück. Im folgenden Snippet definieren wir eine Funktion, die eine Meldung auf der Konsole ausgeben soll.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 29: Definition einer Funktion, die eine Meldung auf der Konsole ausgibt

Eine Funktion beginnt mit den „function“ Keyword gefolgt vom Namen der Funktion, zwei runde Klammern und zwei geschwungene Klammern. Innerhalb der geschwungenen Klammen befindet sich der Funktionsblock, in dem die Befehle und Berechnungen stehen. Man sieht, dass Funktionen nicht mit einem Strichpunkt abgeschlossen werden, da sie ja keine Zuweisung haben wie z.B.: die Deklaration von Variablen.

Um diese Funktion auszuführen, muss man sie mit ihrem Namen aufrufen. Die runden Klammern gehören dazu. Abschlossen wird der Aufruf mit einem Strichpunkt. Wenn man sich die Konsolenausgabe ansieht, sieht man den Text „Hello World“ stehen.

Um eine Funktion interessanter zu machen wäre es doch spannender, jemanden mit seinem Namen zu grüßen. Funktionen haben auch Eingabeparameter. Daher wir können Werte an Funktionen übergeben und diese dann in der Funktion verwenden, um das Verhalten der Funktion zu ändern.

Die neue Funktion hat nun einen Parameter “name“. “name“ ist wie eine Variable, die aber nur im Funktionsblock gültig ist. Daher man kann nicht außerhalb der geschwungenen Klammern auf „name“ zugreifen. Innerhalb der geschwungenen Klammern können wir „name“ verwenden.

Im „console.log“ können wir nun „Hello“ mit der Variable kombinieren. Mit einem „+“ kann man Strings zusammenhängen. Wenn man nun die „greet“ Funktion aufruft, muss man einen Wert für den Parameter „name“ angeben damit wir dieses Verwenden können.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 30: Funktion mit Parameter

In der Konsole werden wir nun „Hello Michael“ sehen. Ein Vorteil von Funktionen ist, dass wir diese immer wieder verwenden können. Wir können nun ein weiters mal „greet“ mit einem anderen Argument aufrufen, um jemand anders zu grüßen.



Abbildung 31: Funktonen wiederverwenden

Nun sehen wir auf der Konsole sowohl „Hello Michael“ als auch „Hello Maria“.

Eine Funktion kann mehrere Parameter haben. Die Parameter werden mit einem Komma getrennt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 32: Funktion mit zwei Parametern

Die Funktion „greet“ wird hier um den Parameter „lastName“ erweitert. Im „console.log“ erweitern wir den String um ein Leerzeichen und den zweiten Parameter um „Hello“, Namen und Naschname auszugeben. Wenn wir nun aber unseren Funktionsaufruf nicht um das „lastName“ Argument erweitern, wird in der Konsole „Hello Michael undefined“ ausgeben. Nicht vorhandene Parameter sind wie deklarierte Variablen „undefined“.

Diese Funktion führt eine Aufgabe aus. Es gibt aber auch Funktionen die Berechnung ausführen und diese zurückgeben. Um in einer Funktion einen Wert an den Aufrufer zurückzugeben wird das „return“ Keyword verwendet.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 33: Funktion mit Rückgabe

Die Funktion „square“ quadriert den übergebenen Parameter und gibt diesen zurück. Da diese Funktion einen Wert zurückgibt, könne wir sie zur Initialisierung einer Variablen verwenden. In der Konsole sollte nun 4 stehen.

In unserm Beispiel können wir uns die Variable auch sparen und den Rückgabewert der Funktion gleich direkt verwenden.

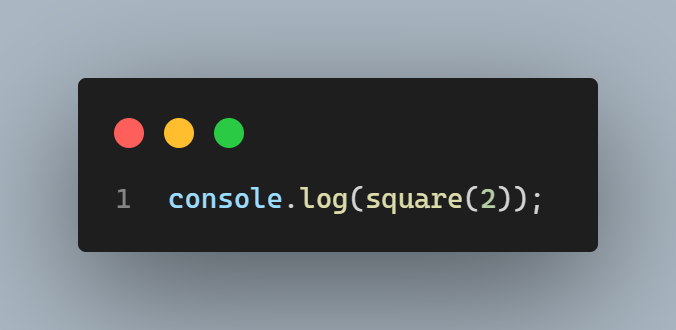


Abbildung 34: Rückgabe Funktion direkt verwenden