JavaScript
für BIC Process Execution

Hochschule Esslingen
Prof. Andreas Rößler

V3: 20.11.2025

Übersicht

- Einführung ins Programmieren mit JavaScript
- Datentypen:
  - Zahlen, Zeichenketten, zusammengesetzte Daten
  - Umwandlung
- Programmierkonzepte
  - Verzweigung, Schleife, Funktion
- Praktisches
  - Schnittstellen zu BIC PE
  - Debugging

Termine

- 02.10.2025
- 17.10.2025
- **21.11.2025, 9 -12 Uhr**

Umfragen

https://particify.hs-esslingen.de/p/48964849

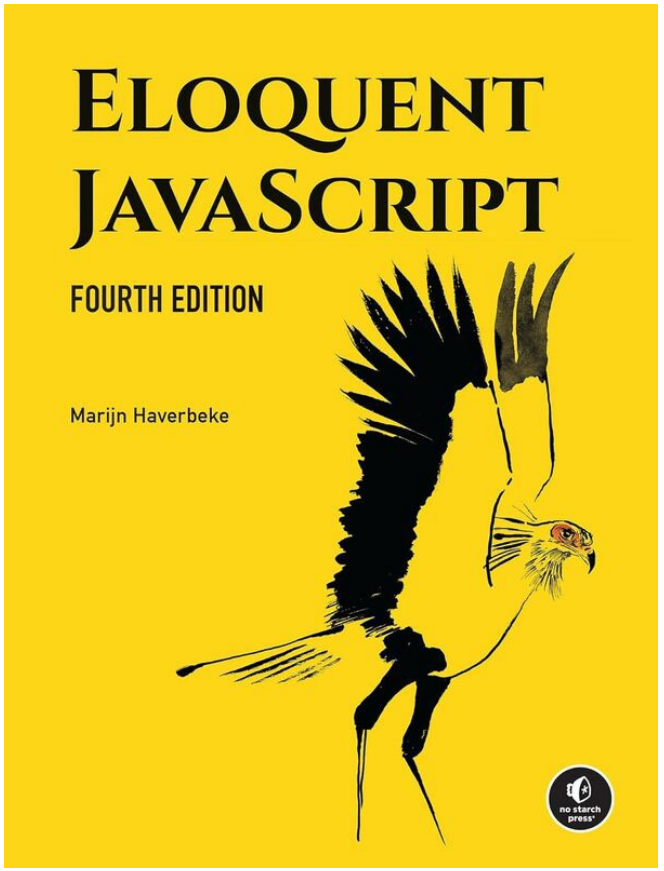
Literatur

Haverbeke

Eloquent JavaScript.

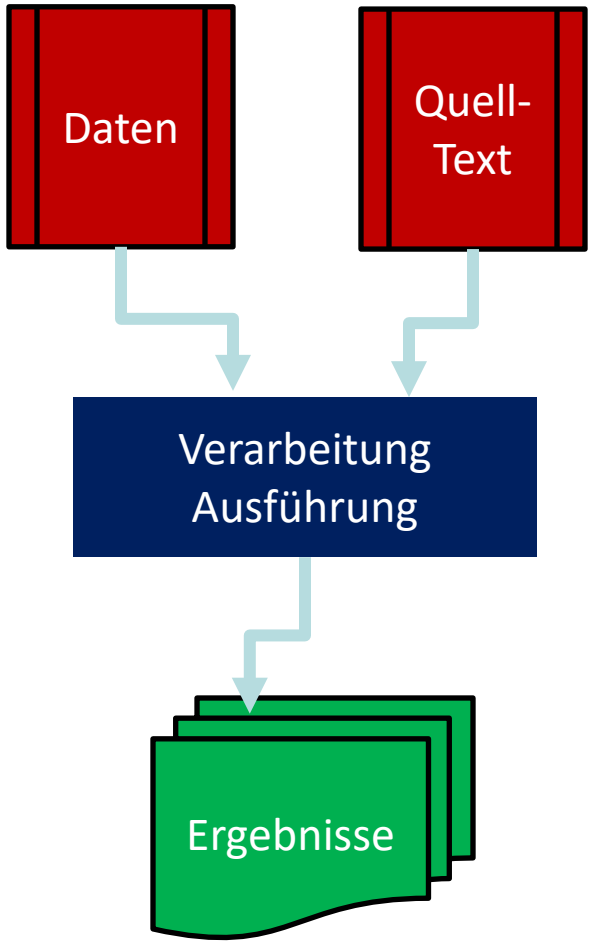
4th Edition, no starch press, 2024

https://eloquentjavascript.net/



Programmieren

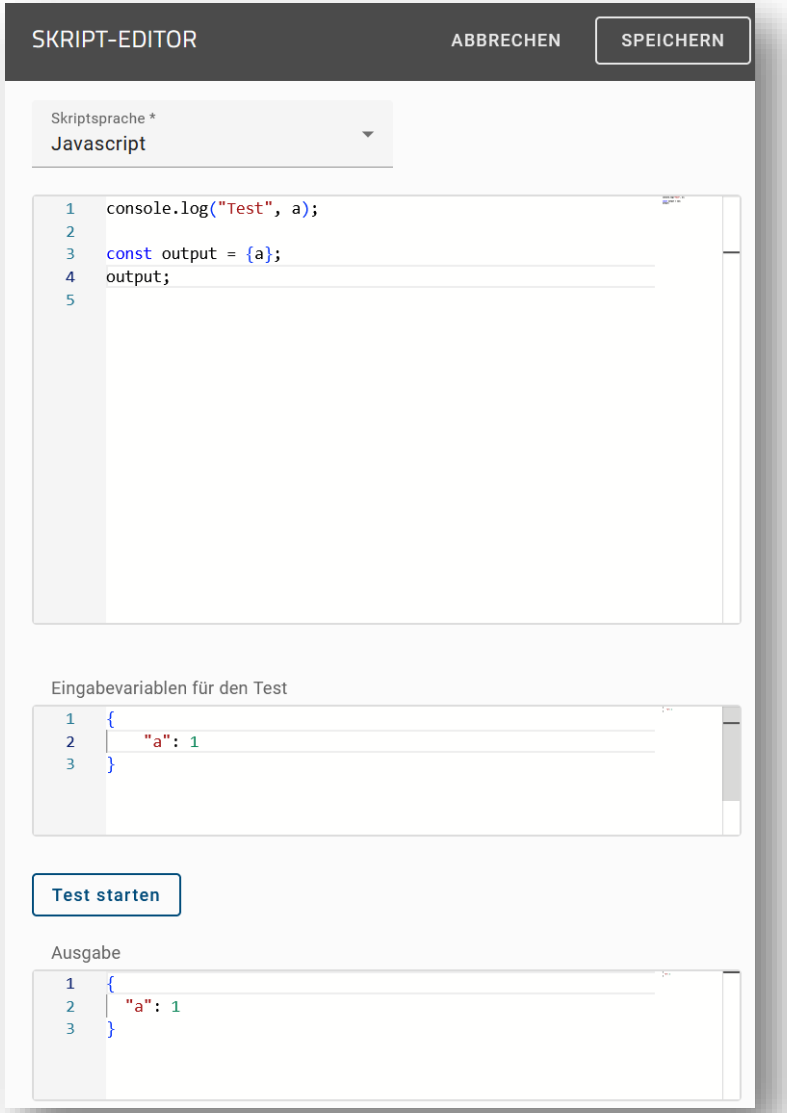
- Niederschreiben eines „Back“-Rezepts:
  - Programm-Ablauf, Algorithmus als Quelltext
- Zutaten und Werkzeuge:
  - Programmiersprache mit
    - vordefinierten Schlüsselworten,
    - Regeln,
    - Funktionen
  - Daten und Datenstrukturen
- Verarbeitung/Ausführung



Testen und Debugging

BIC PE Skript-Editor:

- Test-Möglichkeit mit Ein- und Ausgabe
- ABER:
  - kein console.log
  - relativ langsam (Server-basiert)
  - mehrere Skripte parallel bearbeiten etwas aufwändig



Testen und Debugging

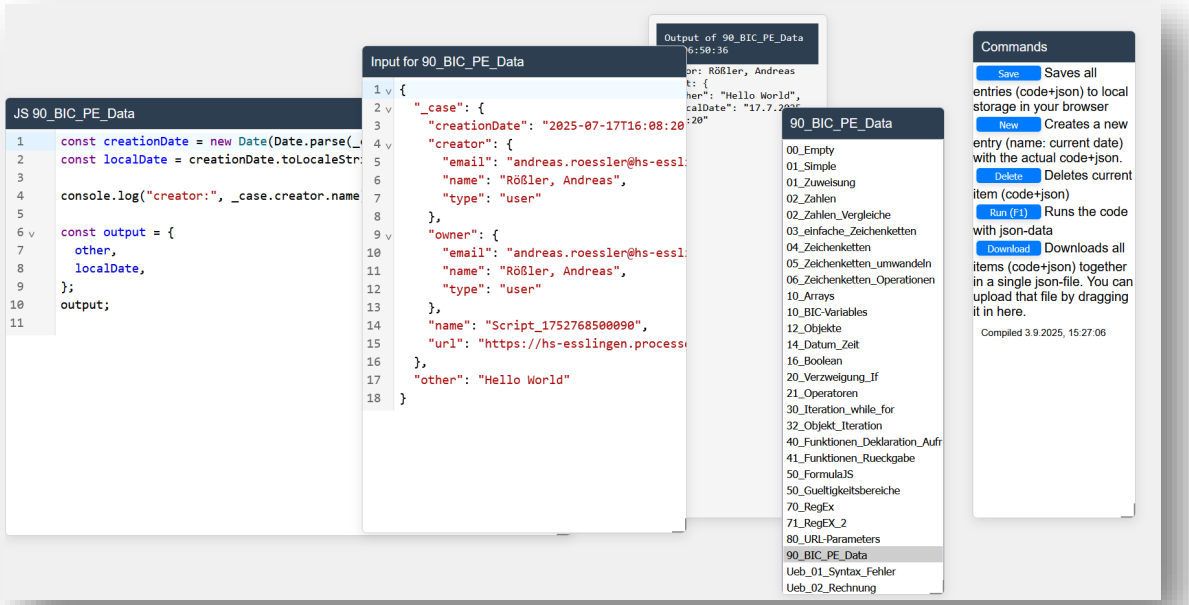
Alternative Test-Möglichkeit:

https://go-hse.github.io/bic-js-env/dist/index.html

Testen mit:

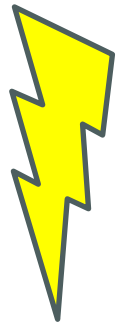
- console.log
- Speicherbare Variablen-Sätze, JSON-Autokorrektur
- JavaScript-Autoformat, Anzeige von JS-Syntax-Fehlern
- Schnell – Ausführung lokal im Browser
- Daten nur lokal im Browser; mit Download
- Open Source

Alpha-Version, kann Fehler enthalten,
mit der Bitte um Nachsicht.



JavaScript Daten

- Grundlegende Datentypen sind in JavaScript
  - Zahlen: `123`, `3.1415`
  - Zeichenketten: `"Hochschule Esslingen"`
  - Wahrheitswerte: `true/false`
- Zusätzlich gibt es:
  - Arrays (Felder)
  - Objekte, u.a. Datum/Zeit
  - Funktionen



In JavaScript werden Komma-Zahlen mit dem Punkt getrennt.

JavaScript Kommentare

- Kommentare werden von JavaScript ignoriert
- Sie werden verwendet, damit Menschen den Quellcode besser verstehen

```
// einzeiliger Kommentar
let name = "test";
/*
Mehrzeiliger Kommentar
*/
```

JavaScript Variablen

- Daten werden im Speicher abgelegt;
  um darauf zugreifen zu können, erhalten sie Namen.
- Die Namen heißen Bezeichner (engl. identifier).
- Die Bezeichner können frei gewählt werden:
  - keine Zahl am Anfang
  - keine Leerzeichen
  - keine Sonderzeichen mit Ausnahme des Unterstrichs _ und des Dollars $
  - Gut sind „sprechende“ Bezeichner: Mehrwertsteuer; MatrikelNummer
  - Verboten: Matrikel Nummer, 1Auto, Mehrwertsteuer%
  - Schlecht: m, x, _, Gasthörer

Deklaration und Initialisierung von Variablen

- Bevor Variablen verwendet werden können müssen sie deklariert (erzeugt/angemeldet) werden.
- Es ist gute Praxis, die Variablen bei der Deklaration auch zu initialisieren, also einen eine Wert zuzuweisen.
- Nicht-initialisierte Variablen haben den Wert `undefined`.

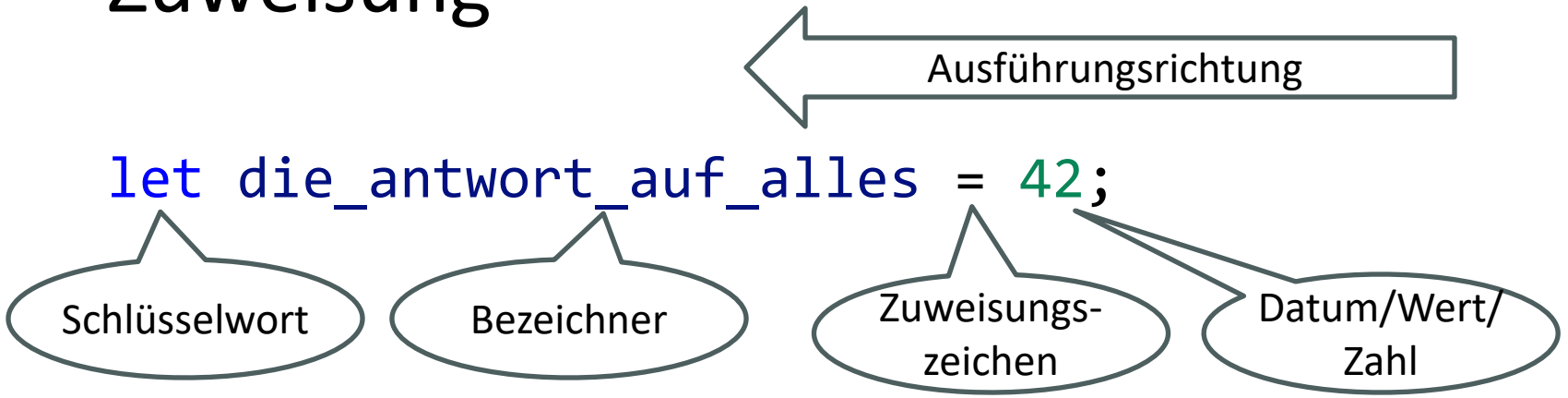
Eingebaute Funktionen verwenden

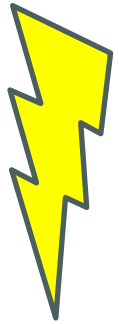
- Funktionen sind vorgefertigte Bausteine, die verwendet werden können.
- Es gibt Funktionen, die nur etwas „tun“.
- Es gibt Funktionen, die ein Ergebnis zurück liefern.

```
// parseInt wird aufgerufen und das Ergebnis in zahl gespeichert
const zahl = parseInt("12em");
// gibt zahl auf der Konsole aus
console.log(zahl);
```

JavaScript Zuweisung

- Zuweisung





Die Anweisung

```
let die_antwort_auf_alles;
```

deklariert die Variable,
und setzt den Wert undefined.

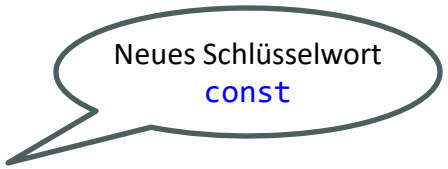
- Schlüsselworte sind durch JavaScript festgelegt.
- Bezeichner können/müssen selbst gewählt werden.
- Das Schlüsselwort `let` deklariert/erzeugt eine Variable (im Speicher veränderbar).
- Danach wird der Variable die Zahl 42 zugewiesen.

Deklaration und Initialisierung von Variablen

```
let die_antwort_auf_alles = 42;  // Deklaration und Initialisierung
console.log("Die Antwort", die_antwort_auf_alles); // Ausgabe - nicht BIC PE

let leere_variable; // Deklaration mit let OHNE Initialisierung
die_antwort_auf_alles = 43; // neue Zuweisung, Veränderung der Variable

// Konstante Variablen; keine weitere Zuweisung
const MwstSatzProzent = 0.19;
const NettoPreisEuro = 10;
const BruttoPreisEuro = NettoPreisEuro * MwstSatzProzent;
console.log("BruttoPreisEuro", BruttoPreisEuro);

// alt:
var name = "Max";
```

Konstante „Variablen“

- In JavaScript können Variablen unveränderbar sein:
  - Sie werden als Konstante bezeichnet.
  - Ein Wert wird bei der Definition gespeichert und kann danach nicht neu zugewiesen werden.
  - Das kann Fehler vermeiden.




```
const x = 2; // Zuweisungen bei der Definition
const y = 1.1;

console.log("Subtraktion", (x - y).toFixed(2));
```

Zuweisung von Werten an Variablen

- Deklarierten Variablen können neue Werte zugewiesen werden.


```
let BruttoPreisEuro = NettoPreisEuro * MwstSatzProzent;
let ODER const
const BruttoPreisEuro = NettoPreisEuro * MwstSatzProzent;
BruttoPreisEuro = NettoPreisEuro * MwstSatzProzent;
```

- Dabei kann u.a. gerechnet werden.

JavaScript Syntax

- Die Syntax umfasst die Regeln, die eingehalten werden müssen, damit der Quelltext vom Computer verstanden wird. Dazu zählen:
  - Sonderzeichen strukturieren den Quelltext.
    - Das Leerzeichen trennt Bezeichner (eigenen Worte), Schlüsselworte, und Operatoren (z.B. +/-).
    - Der Strichpunkt trennt Anweisungen.
    - Das Komma trennt Aufzählungen.
    - Der Punkt trennt Vor- und Nachkommastellen und wird auch bei Objekt-Attributen verwendet.
    - Anführungszeichen kennzeichnen Zeichenketten
    - Auch Klammern (), [], {} haben eine strukturierende Bedeutung.

JavaScript Syntax Fehler

- Wenn das Programm (der Quell-Code) Syntax-Fehler enthält, ist das Programm nicht ausführbar.

```
cnst zahl = 1;
const zahl = 1,1;
const text = 'ABC;

/ ein Kommentar

console.log(text zahl);
console.log(text, zahl;
```

JS: Zahlen

- JavaScript kennt nur einen Datentyp für Zahlen: Number

```
let x = 2; // Zuweisungen
let y = 1.1;

// Ausgabe mit console.log
console.log("Typen", typeof x, typeof y);

console.log("Addition", x + y);
console.log("Subtraktion", (x - y).toFixed(2));
console.log("Multiplikation", x * y);
console.log("Division", (x / y).toFixed(3));

x = 14; // Neue Zuweisung
y = x % 5; // Modulo-Operator: ganzzahliger Rest

console.log("Modulo", y);
```

Kommentar, wird nicht ausgewertet

`console.log` nützlich zur Fehlersuche

Auswertung in der Funktion

`toFixed` wandelt eine Zahl in eine Zeichenkette mit definierter Anzahl Nachkommastellen um

Auswertung in der Anweisung

JS: weitere Operatoren

Arithmetik

- Exponent: 2 **3 == 8
- Negation: -5
- Inkrement/Dekrement: ++i oder --i

Zuweisungen

- =, +=, -=, *=, /=, %=, **=: i += 5;

Übungen

Berechne Deinen Body-Mass-Index:

$$BMI = \frac{Gewicht}{Größe^2}$$

```
const gewicht_kg = 20;
const groesse_m = 1.4;
const bmi = ...;
```

Rechne eine Temperatur in Grad/Celsius in Fahrenheit um:

$$F = (C \times \frac{9}{5}) + 32$$

```
const grad_celsius= 20;
const fahrenheit = ...;
```

siehe: Ueb_01_Rechnung

JS: Vergleichs-Operatoren

Vergleiche (nicht nur) mit Zahlen

- ==, ===, !=, !==, <, >, <=, >=
- und: &&, oder: ||
- Das Ergebnis eines Vergleichs ist ein Wahrheitswert (Boolean).
- Wahrheitswerte werden u.a. für Verzweigungen gebraucht.

```
const x = 10, y = 15;

console.log("x kleiner y:", x < y);
console.log("x gleich y:", x === y);
console.log("x <= y:", x <= y);

console.log("mit und:", x <= 10 && y < 15
console.log("mit oder:", x <= 10 || y < 1
```

JS: Vergleichs-Operatoren

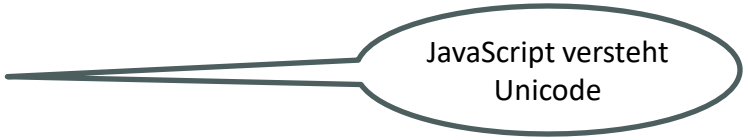
Vergleiche (nicht nur) mit Zahlen

```
const x = 10, y = 15;

console.log("x kleiner y:", x < y);
console.log("x gleich y:", x === y);
console.log("x <= y:", x <= y);

console.log("mit und:", x <= 10 && y < 15);
console.log("mit oder:", x <= 10 || y < 15);
```

JS: Texte/Zeichenketten

- JavaScript verwendet für Texte den Datentyp `string` (deutsch: Zeichenkette)
- Zeichenketten können alle druckbaren Zeichen enthalten:

```
const text = "ABC";
const zahl = "123";

const sonderzeichen = "ÖÄÜ§$%&()?!";
const omega_desertIslandEmoji = "\u03A9 \u{1F3DD}";
const check = "\u2713";
const checkBox = "\u2610";
const checkBoxChk = "\u2611";
const smileys = "\u{1F606} \u{1F60E} 😆 😎 😟";

console.log(sonderzeichen);
console.log(omega_desertIslandEmoji);
console.log(check, checkBox, checkBoxChk);
console.log(smileys);
```



Texte vs. Zahlen

- Texte sind intern auch Zahlen, d.h. sie sind codiert.
- Typisches Codes sind ASCII und Unicode
- ASCII-Tabelle aus https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

ASCII-Zeichentabelle, hexadezimale Nummerierung

Code	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	*NUL*	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
1…	*DLE*	*DC1*	*DC2*	*DC3*	*DC4*	*NAK*	*SYN*	*ETB*	*CAN*	*EM*	*SUB*	*ESC*	*FS*	*GS*	*RS*	*US*
2…	*SP*	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3…	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4…	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5…	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6…	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7…	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	*DEL*

Texte in Zahlen umwandeln

- JavaScript kann Zahlen-Texte in rechenbare Zahlen umwandeln.

Das ist ein Text!

```
const piString = "3.1415";
const pi = parseFloat(piString);
const radius = 10;
const umfang = 2 * pi * radius;
const umfangS1 = piString * 2 * radius;
const umfangS2 = piString + 2 + radius;
console.log("Pi", piString, typeof piString, pi, typeof pi);
console.log("Umfang", umfang);
console.log("Umfang", umfangS1);
console.log("Umfang", umfangS2);

console.log("i", parseInt("101", 2));
console.log("i", parseInt("101", 4));
console.log("i", parseInt("101", 10));
```

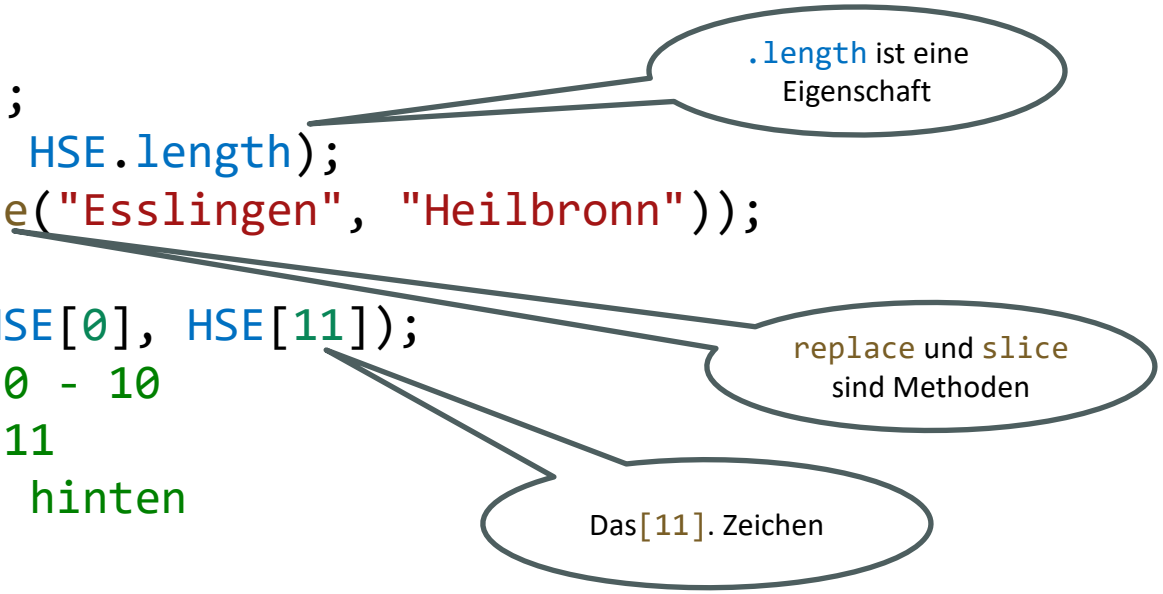
`parseFloat` wandelt in Fließkommazahlen

`parseInt` wandelt in ganze Zahlen, in unterschiedl. Zahlensystemen

Operationen mit Texten

- Texte sind Objekte mit Eigenschaften und Methoden
  - Die length-Eigenschaft enthält die Anzahl der Zeichen.
  - Die replace-Methode ersetzt Teile einer Zeichenkette.
  - Die slice-Methode gibt einen Teil einer Zeichenkette zurück.
  - Mit dem Zugriffs-Operator [] kann auf einzelne Zeichen zugegriffen werden.

```
const HSE = "Hochschule Esslingen";
console.log("Anzahl der Zeichen:", HSE.length);
console.log("Replace:", HSE.replace("Esslingen", "Heilbronn"));

console.log("Einzelne Zeichen:", HSE[0], HSE[11]);
console.log(HSE.slice(0, 10)); // 0 - 10
console.log(HSE.slice(11)); // ab 11
console.log(HSE.slice(-9)); // von hinten
```



Felder - Arrays

Arrays sind Objekte mit Eigenschaften und Methoden

- Array speichern mehrere/viele Daten in einer Struktur.
- Über einen Index kann auf die Daten zugegriffen werden.
- Die length-Eigenschaft enthält die Anzahl der Zeichen.
- Es gibt einige Methoden zur Veränderung von Arrays.
  siehe https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

```
const zahlen = [3, 5, 7, 10];
const strings = ["Andreas", "Rößler", "Hochschule", "Esslingen"];
const gemischt = ["Anna", "Nym", 123456];
console.log(zahlen[2]);
console.log(strings[2]);
console.log(gemischt[2]);
console.log(strings.length);

// Ersetzen; geht auch bei const-array
strings[3] = "Heilbronn";
// Array-Element zu String verknüpfen
console.log(strings.join(", "));
```

```
// suchen; ohne Prüfung, ob Element vorhanden ist
const idx = zahlen.indexOf(10);
zahlen[idx] = 20;

console.log(zahlen.join(", "));

const joined = zahlen.concat(strings);
console.log(joined.join(", "));

const mehrZahlen = [3, 5, 7, 10, [1, 2, 3]];
console.log(mehrZahlen.join("; "));
console.log(mehrZahlen.flat().join(", "));
```

Feld/Array-Funktionen

Es ist praktisch, mit einer Anweisung alle Feldelemente zu bearbeiten.

Dafür gibt es u.a. die Funktionen map, reduce, filter, find, every

```
// Multipliziere alle Zahlen im Array mit 2
const zahlen = [1, 2, 3, 4];
const verdoppelt = zahlen.map(x => x * 2);
console.log(verdoppelt); // [2, 4, 6, 8]

// Summiere alle Zahlen eines Arrays
const werte = [10, 20, 30];
const summe = werte.reduce((acc, val) => acc + val, 0);
console.log(summe); // 60

// Filtere alle geraden Zahlen heraus
const zahlen2 = [1, 2, 3, 4, 5, 6];
const gerade = zahlen2.filter(x => x % 2 === 0);
console.log(gerade); // [2, 4, 6]
```

```
// Finde den ersten Namen, der länger als 4 Buchstaben ist
const namen = ["Tom", "Lisa", "Alexander", "Mia"];
const langerName = namen.find(n => n.length > 4);
console.log(langerName); // "Alexander"

// Prüfe, ob alle Werte größer als 0 sind
const zahlen3 = [3, 5, 8, 1];
const allePositiv = zahlen3.every(x => x > 0);
console.log(allePositiv); // true
```

Objekte - JSON

Objekte speichern Schlüssel-Wert-Paar.

- Der Schlüssel ist eine Zeichenkette.
- Der Wert kann jeden Datentyp haben, also Zeichenketten, Zahlen, Arrays oder ein Objekt.
- JSON steht für JavaScript Object Notation.

```
const zahlen = [3, 5, 7, 10];
const strings = ["Andreas", "Rößler", "Hochschule", "Esslingen"];
const gemischt = ["Anna", "Nym", 123456];
const dasObjekt = {
    zahlen: zahlen,
    strings: strings,
    alter: 25,
    vorname: "Anna",
    nachname: "Nym",
    adresse: {
        strasse: "Müllerstraße 15",
        ort: "Buxtehude",
    },
};
```

```
console.log(dasObjekt.alter);
console.log(dasObjekt.zahlen[1]);
// Zugriff auf Werte über den Schlüssel
dasObjekt.telefonNr = "0123-45678";
dasObjekt["telefonNr"] = "0123-45678";

const attribut = "telefonNr";
dasObjekt[attribut] = "0123-45678";
```

Zugriffs-Methoden

Objekt in JSON-Notation

Objekte in Arrays

Gleichartige Objekte können gut in Arrays gespeichert und verarbeitet werden:

```
// Beispiel 1: Finde einen bestimmten Studierenden nach Matrikelnummer
const studenten = [
    { name: "Anna", matrikel: 101 },
    { name: "Ben", matrikel: 102 },
    { name: "Clara", matrikel: 103 }
];
const gesucht = studenten.find(s => s.matrikel === 102);

// Beispiel 2: Filtere alle Studierenden mit mehr als 30 ECTS-Punkten
const studenten2 = [
    { name: "Anna", ects: 28 },
    { name: "Ben", ects: 42 },
    { name: "Clara", ects: 36 }
];
const fortgeschrittene = studenten2.filter(s => s.ects > 30);

// Beispiel 3: Erzeuge ein Array mit nur den Namen der Dozierenden
const dozierende = [
    { name: "Dr. Meyer", fach: "Mathematik" },
    { name: "Prof. Schulz", fach: "Informatik" }
];
const namen = dozierende.map(d => d.name);
```

```
// Beispiel 4: Berechne die durchschnittliche Note aller Prüfungen
const pruef = [
    { fach: "Mathe", note: 2.0 },
    { fach: "Physik", note: 1.7 },
    { fach: "Informatik", note: 1.3 }
];
const avg = pruef.reduce((sum, p) => sum + p.note, 0) / pruef.length;


// Beispiel 5: Erstelle eine Liste der Kursnamen mit ECTS-Werten
const kurse = [
    { name: "Programmieren", ects: 5 },
    { name: "Mathe 1", ects: 6 },
    { name: "Ethik", ects: 3 }
];
const kursListe = kurse.map(k => `${k.name} (${k.ects} ECTS)`);
```

Datum und Uhrzeit

JavaScript liefert das Date-Objekt

- Das Date-Objekt speichert intern Datum/Uhrzeit als Zeitstempel: Millisekunden seit 1.1.1970.
- Ein Date-Objekt kann mit dem Aufruf `new Date()` erzeugt werden; dabei können unterschiedliche Parameter verwendet werden.
- Das Date-Objekt hat einige Methoden, siehe
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

```
const d = new Date(); // das aktuelle Datum und Uhrzeit
console.log(new Date(0).toLocaleString()); // 1 Parameter: Zeit in ms
// 2 oder mehr Parameter: Y/M-1/D/H/M/S/ms
console.log(new Date(2025, 8, 30, 8, 15).toLocaleString());
const yyyy = d.getFullYear().toString();
const mm = (d.getMonth() + 1).toString().padStart(2, "0");
const dd = d.getDate().toString().padStart(2, "0");
const hh = d.getHours().toString().padStart(2, "0");
const mi = d.getMinutes().toString().padStart(2, "0");
const sc = d.getSeconds().toString().padStart(2, "0");
const wd = d.getDay();

const WD = ["So.", "Mo.", "Di.", "Mi.", "Do.", "Fr.", "Sa."];
console.log(`${WD[wd]}, ${dd}.${mm}.${yyyy} ${hh}:${mi}:${sc}`);
```

```
const nextWeek = new Date(d);
nextWeek.setDate(nextWeek.getDate() + 7); // rechnen mit Datum
console.log("Local", nextWeek.toLocaleString());
console.log("ISO", nextWeek.toISOString());

// Datum von Zeichenkette umwandeln
const stringDateLocal = "3.9.2025, 08:14:33";
// Date.parse liefert Zeitstempel (in ms)
const parsedLocal = new Date(Date.parse(stringDateLocal));
const stringDateISO = "2025-09-03T06:19:34.046Z";
const parsedISO = new Date(Date.parse(stringDateISO));

console.log(parsedLocal.toLocaleString(), "---",
parsedISO.toString());
```

Wahrheitswerte/Booleans

Um Zustände zu speichern, die wahr oder falsch sein können wird der Datentyp Boolean eingesetzt.

- Schlüsselworte für wahr und falsch in JavaScript: `true`, `false`
- Ergebnis von Vergleichsoperatoren: ==, ===, !=, <, >, <=, >=
- Wahrheitswerte können verknüpft werden: und &&, oder ||

Konjunktion (UND)

∧	0	1
0	0	0
1	0	1

Disjunktion (ODER)

∨	0	1
0	0	1
1	1	1

Negation (NICHT)

	¬
0	1
1	0

```
const d = new Date(),
    sc = d.getSeconds(),
    mi = d.getMinutes();

// Wahrheitswerte
const isEven = sc % 2 == 0;

console.log(`Sekunde ${sc} gerade? ${isEven}`);
console.log(`${sc} UND ${mi} gerade? ${sc % 2 == 0 && mi % 2 == 0}`);
console.log(`${sc} ODER ${mi} gerade? ${sc % 2 == 0 || mi % 2 == 0}`);
```

Modulo-Operator

```
const isEven = sc % 2 == 0;
```

Zuweisungs-Operator

Vergleichs-Operator

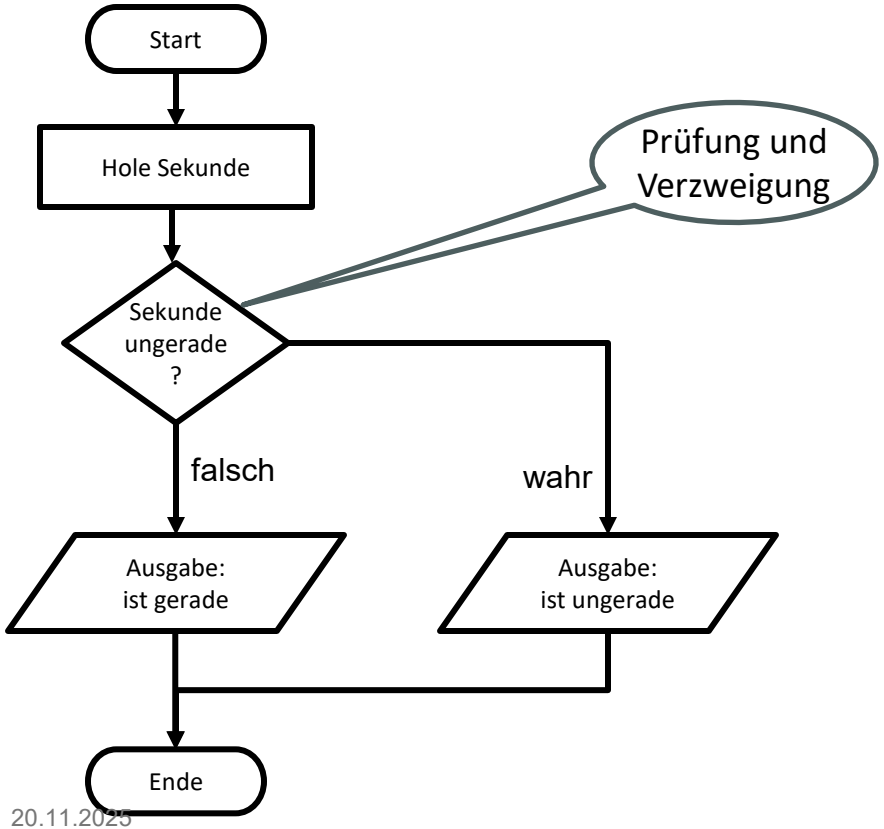
UND

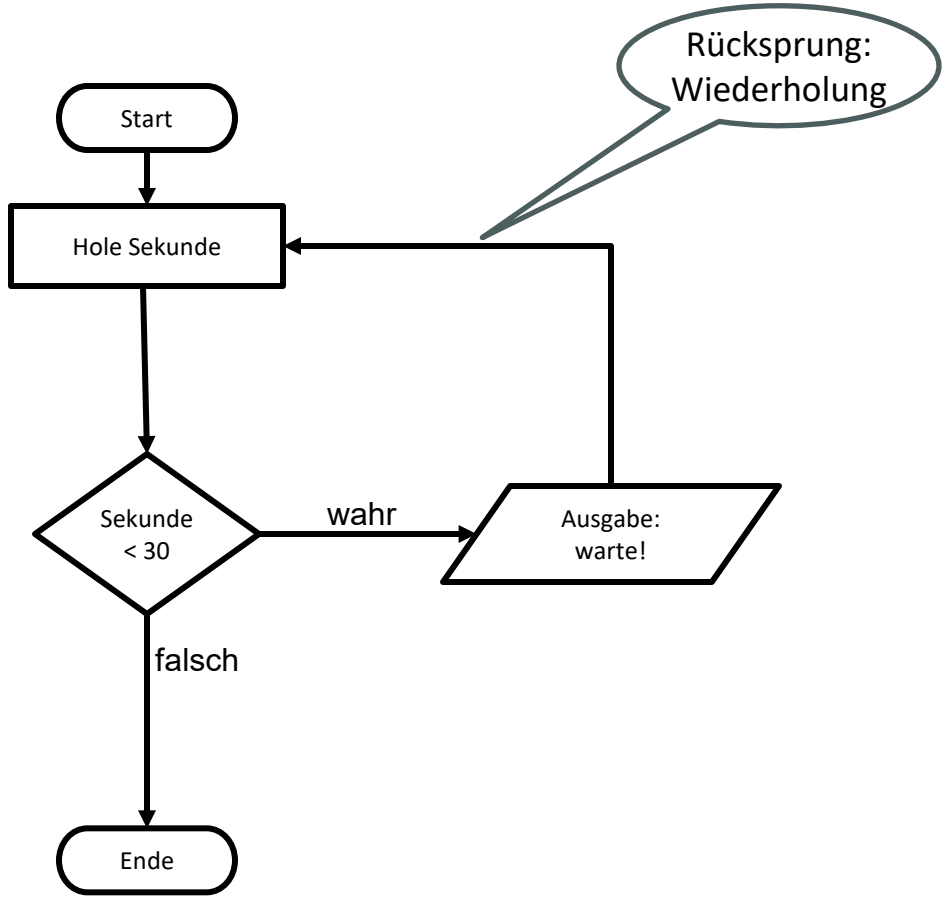
ODER

Kontrolle des Programmflusses

In allen höheren Programmiersprachen gibt es zwei Konstrukte:

- Verzweigungen erlauben unterschiedliche Abläufe.
- Schleifen/Wiederholungen erlauben mehrfache Abläufe.





Verzweigungen

Eine Verzweigung steuert, welche Anweisungen ausgeführt werden. Sie besteht aus

- Dem Schlüsselwort `if`.
- Einer Bedingung in runden Klammern (), die einen Wahrheitswert ergibt.
- Eine Anweisungsliste in geschweiften Klammern {}, die ausgeführt wird, falls die Bedingung wahr ist.
- Optional: einem oder mehrere `else if` Zweige, mit weiteren Bedingungen/Anweisungen.
- Optional: einem `else` Zweig, der ausgeführt wird falls die Bedingung falsch ist.

```
const sc = new Date().getSeconds();
if (sc % 2 == 0) {
    console.log(`${sc} ist gerade`);
} else {
    console.log(`${sc} ist ungerade`);
}
```

```
if (sc < 15) {
    console.log(`${sc} - eine viertel Minute`);
} else if (sc < 30) {
    console.log(`${sc} - eine halbe Minute`);
} else if (sc < 45) {
    console.log(`${sc} - eine dreiviertel Minute`);
} else {
    console.log(`${sc} - eine Minute`);
}
```

Schleifen/Iterationen

JavaScript kennt verschiedene Iterations-Möglichkeiten:

- die klassischen for-Zählschleife und while-Schleife
- Iteration über Objekt-Attribute: for-in
- Iteration über Array-Element: for-of

```
// while
let i = 0; // Zählvariable
while (i < 10) {
    // Bedingung
    console.log(`wa: ${i}`);
    i = i + 2; // Veränderung der Zählvariable
}
while (i > 0) {
    console.log(`wb: ${i}`);
    i = i - 2;
}
```

```
// for
for (let i = 0; i < 10; i = i + 2) {
    console.log(`fa: ${i}`);
}
for (let i = 10; i > 3; i = i - 3) {
    console.log(`fb: ${i}`);
}
for (let i = 0; i < 10; ++i) {
    // ++i entspricht i = i + 1
    if (i % 2 == 0) continue; // geht zum nächsten Schritt
    console.log(`fc: ${i}`);
}
for (let i = 1; i < 10; ++i) {
    if (i % 4 == 0) break; // beende die Schleife
    console.log(`fd: ${i}`);
}
```

Iterationen über Objekte und Arrays

JavaScript kennt Iterations-Möglichkeiten für Objekte und Arrays:

- Iteration über Objekt-Attribute: for-in (Sortierung nicht garantiert)
- Iteration über Array-Element: for-of

```
const dasObjekt = {
    alter: 25,
    vorname: "Anna",
    nachname: "Nym",
    adresse: {},
};

for (const k in dasObjekt) {
    console.log(`(in) dasObjekt[${k}] ist ${dasObjekt[k]}`);
}
```

```
// Objekt.keys liefert Array mit allen Schlüsseln
const keys = Object.keys(dasObjekt).sort();
for (const k of keys) {
    console.log(`(of) dasObjekt[${k}] ist ${dasObjekt[k]}`);
}

const WD = ["So.", "Mo.", "Di.", "Mi.", "Do.", "Fr.", "Sa."];
for (const k of WD) {
    console.log(`Wochentag ${k}`);
}

for (let i = 0; i < WD.length; ++i) {
    console.log(`Wochentag ${i}: ${WD[i]}`);
}
```

Funktionen

JavaScript bringt sehr viele Funktionen mit, die aufgerufen werden können.

Eine kleine Auswahl:

- Allgemein: parseInt(), parseFloat();
- Mathematik: Math.round(), Math.pow(), Math.sqrt();
- Zeichenketten: indexOf(), slice(), replace();
- Arrays: push(), pop(), shift(), indexOf();
- Datum und Uhrzeit: new Date(), parse(), toLocaleString();
- JSON.stringify, JSON.parse();
- Web: fetch();

Eigene Funktionen

Funktionen sind wichtig, um eigene Programme zu strukturieren:

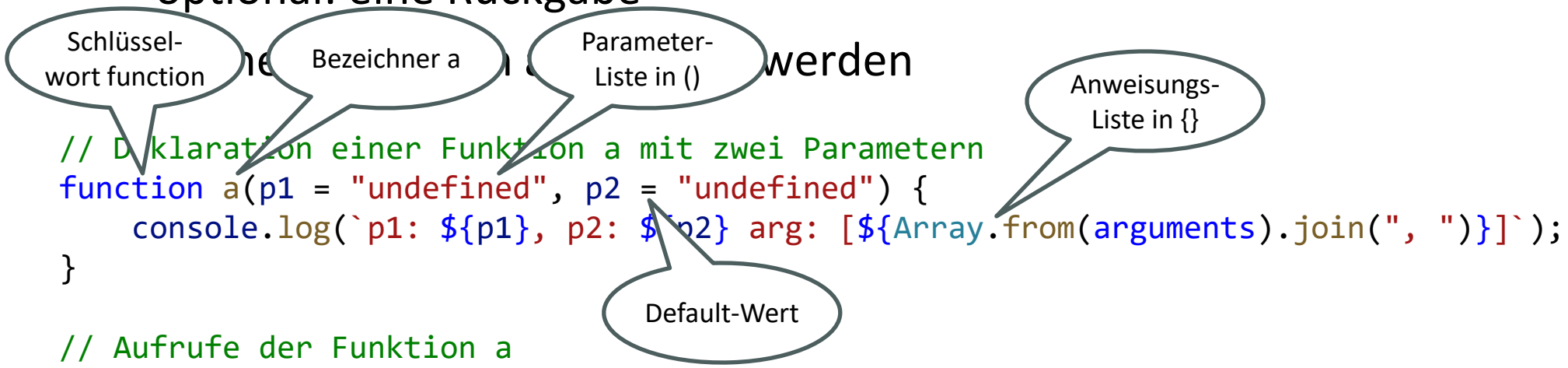
- Vermeidung von Anweisungsblöcken, die sich wiederholen
- kürzerer Code
- besser verständlich
- besser wartbar

Aber: im Kontext von BIC PE sind die JS-Skripte eher kurz;

Eigene Funktionen

Funktionen haben :

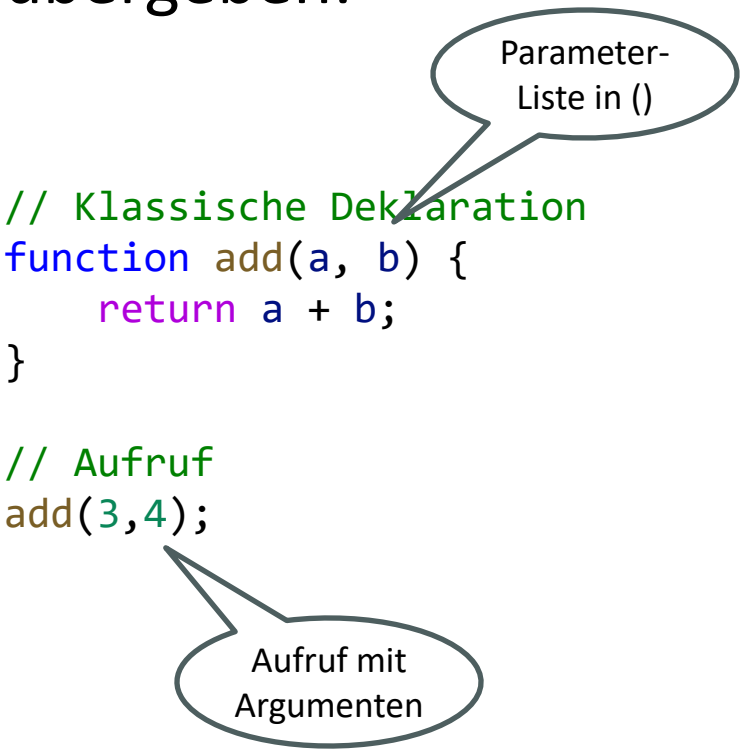
- i.d.R. einen Bezeichner
- eine Parameter-Liste in ()
- eine Anweisungsliste
- optional: eine Rückgabe




```
// Deklaration einer Funktion a mit zwei Parametern
function a(p1 = "undefined", p2 = "undefined") {
    console.log(`p1: ${p1}, p2: ${p2} arg: [${Array.from(arguments).join(", ")}]`);
}

// Aufrufe der Funktion a
a();
a(1);
a(1, 2);
a(1, 2, 3, 4);
```

Parameter und Argumente

Parameter sind Teil der Deklaration, Argumente werden bei Aufruf übergeben:

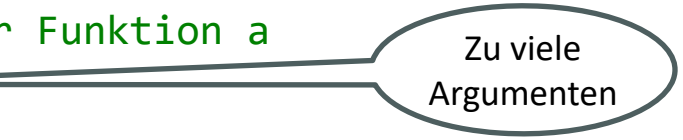


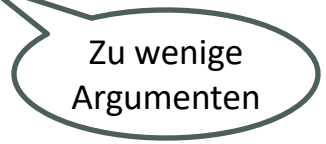
Parameter und Argumente

JavaScript kann sehr variabel mit Argumenten umgehen:

- Fehlende Argumente sind `undefined`
- Überzählige Argumente landen in `arguments`

```
// Deklaration einer Funktion a mit zwei Parametern
function a(p1 = "undefined", p2 = "undefined") {
    console.log(`p1: ${p1}, p2: ${p2} arg: [${Array.from(arguments).join(", ")}]`);
}

// Aufrufe der Funktion a
a();
a(1);
a(1, 2);
a(1, 2, 3, 4);
```





Erstellen von Funktionen

Varianten:

```
// Klassische Deklaration
function add(a, b) {
    return a + b;
}

// Funktionsausdruck
const add = function(a, b) {
    return a + b;
};

// Arrow Function
const add = (a, b) => a + b;
```

Erstellen von Funktionen

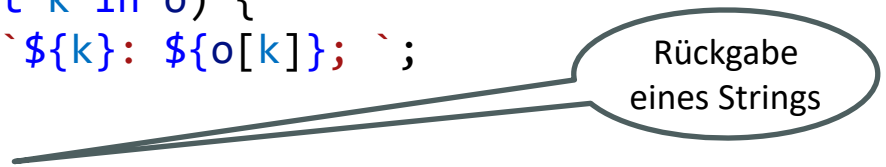
Verarbeitung von Arrays, z.B. aus einem Multi-Options-Formular-Feld:

```
const calcFunctions = {
    "SWB": () => { console.log("Softwaretechnik") },
    "TIB": () => { console.log("Technische Informatik") },
}

const keys = ["SWB", "TIB"];
for (const key of keys) {
    calcFunctions[key]();
}
```

Funktionen mit Rückgabe

Rückgabe von Werten/Objekten mit return:

```
// Deklaration einer Funktion a mit zwei Parametern
function createObject(name, alter) {
    if (typeof alter === "string") {
        alter = parseInt(alter, 10);
    }
    return { name, alter };
}
// Umwandlung Objekt in String
function o2s(o) {
    let s = "";
    for (const k in o) {
        s += `${k}: ${o[k]}; `;
    }
    return s;
}
// Aufrufe der Funktion a und o2s
console.log(o2s(createObject("Ida", 7)));
console.log(o2s(createObject("Hans", "73")));
console.log(JSON.stringify(createObject("Hans", "73"), null, 2));
```



Gültigkeit von Variablen

Variablen sind jeweils lokal oder global gültig.


```
let x = 17;
console.log("globales x", x);

function GFUnktion() {
    console.log("globales x in GFunktion", x);
}

function Funktion(p) {
    // eigener Gültigkeitsbereich
    let x = p * 2; // lokale Variable
    console.log("lokales x", x);
    if (p > 10) {
        // innerer Gültigkeitsbereich
        let x = 100;
        console.log("sehr lokales x", x);
    }
    return x;
}
GFUnktion();
console.log("Ergebnis", Funktion(15));
```

Reguläre Ausdrücke

Für das Umwandeln von Zeichenketten in Datenstrukturen sind reguläre Ausdrück (Regex) nützlich.

Zum Aufbau hilft z.B. https://regex101.com/

```
function parseDate(str) {
  const m = str.match(/^(\d{1,2})\/(\d{1,2})\/(\d{4})$/);
  return m ? new Date(m[3], m[2] - 1, m[1]) : null;
}

const validdates = ["24/12/2015", "7/8/1975"];
```