ECMA SCRIPT 6+

BLOCKSCOPEVARIABLEN UND KONSTANTEN

BLOCK SCOPE MIT LET

```
for (let i = 0; i < a.length; i++) {
    let x = a[i];
    ...
}


let callbacks = [];

for (let i = 0; i <= 2; i++) {
    callbacks[i] = () => i * 2;
}

// callbacks[0]() === 0
// callbacks[1]() === 2
// callbacks[2]() === 4
```

BLOCK SCOPES MIT { ... }

```
{
    function foo () { return 1 }         // foo() === 1
    {
        function foo () { return 2 }     // foo() === 2
    }
                                         // foo() === 1
}
```

```
ECMAScript 5:
(function () {
    var foo = function () { return 1; }
    foo() === 1;
    (function () {
        var foo = function () { return 2; }
        foo() === 2;
    })();
    foo() === 1;
})();
```

CONST

```
const PI = 3.141593

PI > 3.0
```

```
ECMAScript 5:
Object.defineProperty(typeof global === "object" ? global : window, "PI", {
    value:        3.141593,
    enumerable:   true,
    writable:     false,
    configurable: false
})
PI > 3.0;
```

THEMENBLOCK FUNKTIONEN

SPREAD- UND RESTOPERATOR

- Defaultparameter in Funktionen
- Funktionsparameter, Rest-Parameter
- Arrow Funktionen vs. Function-Funktionen

DEFAULT PARAMETER, REST-PARAMETER, SPREAD OPERATOR

DEFAULT PARAMETER

```
function f (x, y = 7, z = 42) {
    return x + y + z
}
```

```
function f(x){
    let x = x || 42;
}

f() // x=undefined, y=7; z=42
```

REST PARAMETER

```
function sum(...theArgs) {
  return theArgs.reduce((previous, current) => {
    return previous + current;
  });
}

console.log(sum(1, 2, 3));
// expected output: 6

console.log(sum(1, 2, 3, 4));
// expected output: 10
```

REST PARAMETER

- Rest Parameter bilden ein Array.
- Methoden wie sort, map, forEach oder pop können direkt angewendet werden.
- Das arguments Objekt ist kein echtes Array.
- Das arguments Objekt hat zusätzliche, spezielle Funktionalität (wie die calleeEigenschaft).

SPREAD OPERATOR

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ] // [ 1, 2, "hello", true, 7 ]

function f (x, y, ...a) {
    return (x + y) * a.length   // f(1, 2, ...params) === 9
}


var str = "foo"
var chars = [ ...str ]          // [ "f", "o", "o" ]
```

ARROW FUNCTIONS

```
( )  =>  {}
```

ARROW FUNCTIONS

- Der Ausdruck einer Pfeilfunktion ist kürzer als ein Funktionsausdruck
- Kein eigenes this, arguments, super, oder new.target.
- Sie können nicht als/in Konstruktoren verwendet werden.

```
(param1, param2, …, paramN) => expression

// gleich zu: => { return expression; }
```

```
(singleParam) => { statements }
(oneParam, twoParam, …, paramN) => { statements }
```

Klammern sind optional, wenn nur ein Parametername vorhanden ist:

```
singleParam => { statements }
```

Das ist also auch möglich:

```
singleParam => returnStatement
```

```
// Die Parameterliste für eine parameterlose Funktion
muss mit einem Klammernpaar geschrieben werden:

() => { statements }
```

Der Body kann eingeklammert werden, um ein Objektliteral zurück zu geben:

```
params => ({foo: bar})
```

Rest-Parameter und Default-Parameter:

```
(oneParam, oneParam, ...rest) => { statements }

(param1 = defaultValue) => { statements }
```

TEMPLATE LITERALS

- String interpolation
- Custom interpolation
- Raw string access

STRING INTERPOLATION

```
var customer = { name: "Foo" }

var card = { amount: 7, product: "Bar", unitprice: 42 }

var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`
```

CUSTOM INTERPOLATION

```
get (`http://example.com/foo?bar=${bar + baz}&quux=${quux}`)
```

CLASSES

CLASSES

```
class MyClass {
   constructor () {}

   getProperty() {}
   setProperty() {}
}
```

CLASSES

```
class ChildClass extends parentClass {
   constructor () {}

   getProperty() {}
   setProperty() {}
}
```

KLASSEN, KONSTRUKTOR,

```
class Shape {
    constructor (id, x, y) {
        this._id = id
        this._move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
    getId () {}
    setMove(){}
}
```

```
ECMAScript 5 – syntactic sugar: reduced | traditional
var Shape = function (id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move = function (x, y) {
    this.x = x;
    this.y = y;
};
```

VERERBUNG, ELTERNKONSTRUKTOR

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y);
        this.width  = width;
        this.height = height;
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y);
        this.radius = radius;
    }
}
```

DESTRUCTURING VON ARRAYS UND OBJEKTEN

DESTRUCTURING

- Die destrukturierende Zuweisung ermöglicht es, Daten aus Arrays oder Objekten zu extrahieren

- Die Syntax ist der Konstruktion von Array- und Objekt-Literalen nachempfunden.

- Destructuring ist "fail-soft", ähnlich wie Standardobjekte, die nach `foo["bar"]`, schauen, und ggf. nur ein `undefined` liefern.

OBJECT AND ARRAY MATCHING

```
var a, b, rest;
[a, b] = [10, 20];                        // a === 10
                                          // b === 20


[a, b, ...rest] = [10, 20, 30, 40, 50];   // a === 10
                                          // b === 20
                                          // rest === [30, 40, 50]

{ a, b } = { c: 10, d: 20 };              // a === 10
                                          // b === 20
```

OBJECT AND ARRAY MATCHING, DEFAULT VALUES

```
// Fail-soft destructuring
var [a] = [];                // a === undefined;


// Fail-soft destructuring with defaults
var [a = 1] = [];            // a === 1;
```

Destrukturierung in der Parameterliste:

```
var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c;

f(); // 6
```

SPREAD- UND RESTOPERATOR

- Destructuring von Arrays
- Destructuring von Objekten

MODULE

MODULE

- Language-level support for modules for component definition.

- Codifies patterns from popular JavaScript module loaders (AMD, CommonJS).

- Runtime behaviour defined by a host-defined default loader.

- Implicitly async model — no code executes until requested modules are available and processed.

MODULES

```
// Modul: lib/math.js
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;


// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));


// otherApp.js
import {sum, pi} from "lib/math";
alert("2π = " + sum(pi, pi));
```

VERSCHIEDENE EXPORT MÖGLICHKEITEN

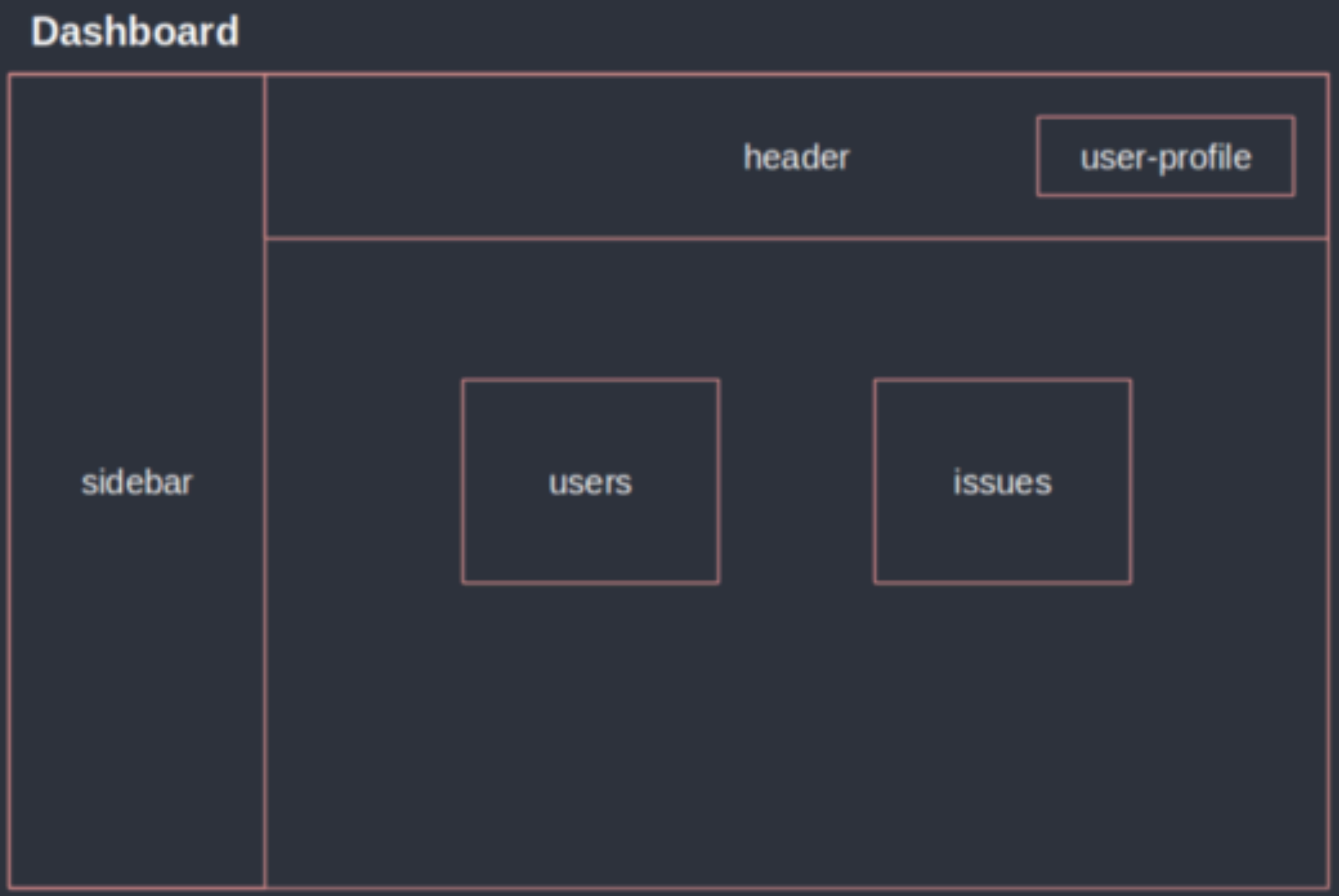
```
export { name1, name2, …, nameN };
export function FunctionName(){...}

export class ClassName {...}
export default class ClassName {...}

...

export * from …;
export { name1, name2, ..., nameN } from ...;
export { import1 as name1, import2 as name2, ..., nameN } from
…;
export { default } from ...;
```

EINE APPLIKATION MIT MODULEN ENTWERFEN



Dashboard

sidebar

header

user-profile

users

issues

SCHRITT 1 - ENTWURF

Ein guter Entwurf erspart viel Zeit und Kopfschmerzen. Ein Entwurf muss nicht perfekt sein, er sollte aber die Richtung weisen.

Entwurf für eine Softwarearchitektur:

Components: users.js, user–profile.js, issues.js
Layouts: header.js, sidebar.js
Dashboard: dashboard.js

Alle Komponenten werden in dashboard.js geladen. Das Dashboard wird dann über index.js initiiert.

SCHRITT 1 - ENTWURF

Das Layout brauchen wir nur einmal, zum Beispiel als statisches Template. Der Inhalt darin mag sich ändern, aber die Sidebar und der Header werden gleich bleiben.

Der Komponenten Ordner ist für allgemeine Komponenten, die wiederverwendbar sind oder sein sollen.

SCHRITT 2 - ORDNERSTRUKTUR

```
root
|- dashboard
|   |- dashboard.js
|
|- components
|   |- issues.js
|   |- user.js
|   |- userprofile.js
|
|- layouts
|   |- header.js
|   |- sidebar.js
|
|- snippets
|   |- user-data.html | .jade | .ejs
|- index.html
|- index.js ( entry point )
```

SCHRITT 3 — IMPLEMENTATION

Komponenten bauen: Jede Komponente ist eine Klasse
Eine Methode zeigt das Laden einer Komponente an.

```
class Users {
  loadUsers() {
    console.log('Users component is loaded...')
  }
  buildHtml(){
    // load Data
    // load template
    // merge both and return
  }
}
export { Users };
```

SCHRITT 3 — IMPLEMENTATION

```
import { UserProfile } from '../components/users-profile.js';

class Header {
  loadHeader() {
    // Creata a new instance
    const userProfile = new UserProfile();

    // Invoke the method (component)
    userProfile.loadUserProfile();

    // Output loading status
    console.log('Header component is loaded...')
  }
}

export { Header };
```

SCHRITT 3 – IMPLEMENTATION

```
// From component folder
import { Users } from '../components/users.js';
import { Issues } from '../components/issues.js';

// From layout folder
import { Header } from '../layouts/header.js';
import { Sidebar } from '../layouts/sidebar.js';

class Dashboard {
  loadDashboard(){

    // Create new instances
    const users = new Users();
    const issues = new Issues();
    const header = new Header();
    const sidebar = new Sidebar();

   function addToDashboard(users.buildHtml(), '#user-layout-id'){ ... }

    console.log('Dashboard component is loaded');
  }
}

export { Dashboard }
```

SCHRITT 3 — IMPLEMENTATION

```
// index.js:

import { Dashboard } from './dashboard/dashboard.js';

const dashboard = new Dashboard();
dashboard.loadDashboard();
```

https://www.freecodecamp.org/news/how-to-use-es6-modules-and-why-theyre-important-a9b20b480773/

PROXIES

PROXY

```
let target = {
  foo: "Welcome, foo"
}

let proxy = new Proxy(target, {
  get(receiver, name) {

    if (name in receiver) {
      value = receiver[name];
    } else {
      value = `Hello, ${name}`;
    }

    return value;
  }
})

// proxy.foo === "Welcome, foo"
// proxy.world === "Hello, world"
```

NEUE METHODEN FÜR STANDARDOBJEKTE IN ES2015

NEUE ARRAY METHODEN

ARRAY METHODS

```
Array
    .from()
    .isArray()
    .of()

Array.prototype
    .concat()
    .copyWithin()
    .entries()
    .every()
    .fill()
    .filter()
    .find()
    .findIndex()
    .flat()
    .flatMap()
    .forEach()
    .includes()
    .indexOf()
    .join()
    .keys()
    .lastIndexOf()
    .map()
    .pop()
    .push()
    .reduce()
    .reduceRight()
    .reverse()
    .shift()
    .slice()
    .some()
    .sort()
    .splice()
    .toLocaleString()
    .toSource()
    .toString()
    .unshift()
    .values()
```

OBJECT TO ARRAY

```
const arrayLikeObject = { length:2, 0:'a', 1:'b' };

for (const x of arrayLikeObject) {       // TypeError
    console.log(x);
}

const arr = Array.from(arrayLikeObject);
for (const x of arr) {                   // OK, iterable
    console.log(x);
}

// Output:
// a
// b
```

MAP()

```
const spanElements = document.querySelectorAll('span.name');

const names1 = Array.prototype.map.call(
   spanElements,
   s => s.textContent
);

// Array.from():
const names2 = Array.from(spanElements, s => s.textContent);
```

ARRAY CREATING

```
Array.of(item_0, item_1, ···)
```

creates an Array whose elements are item_0, item_1, etc.

ARRAY.FIND(), ARRAY.FINDINDEX()

```
let arr = [ 1, 3, 4, 2 ]

arr.find(x => x > 3)      // 4
arr.findIndex(x => x > 3) // 2
```

```
ECMAScript 5
var arr = [ 1, 3, 4, 2 ]
arr.filter( function (x) { return x > 3; } )[0]; // 4
```

TYPED ARRAYS

```
// create a TypedArray with a size in bytes
const typedArray1 = new Int8Array(8);
typedArray1[0] = 32;

const typedArray2 = new Int8Array(typedArray1);
typedArray2[1] = 42;

console.log(typedArray1);
// expected output: Int8Array [32, 0, 0, 0, 0, 0, 0, 0]

console.log(typedArray2);
// expected output: Int8Array [32, 42, 0, 0, 0, 0, 0, 0]
```

TYPED ARRAYS

```
Int8Array();
Uint8Array();
Uint8ClampedArray();
Int16Array();
Uint16Array();
Int32Array();
Uint32Array();
Float32Array();
Float64Array();
BigInt64Array();
BigUint64Array();
```

NEUE METHODEN VON OBJECT

NEW OBJECT METHOD

```
Object
    .assign()
    .create()
    .defineProperties()
    .defineProperty()
    .entries()
    .freeze()
    .fromEntries()
    .getOwnPropertyDescriptor()
    .getOwnPropertyDescriptors(
)
    .getOwnPropertyNames()
    .getOwnPropertySymbols()
    .getPrototypeOf()
    .is()
    .isExtensible()
    .isFrozen()
    .isSealed()
    .keys()
```

```
    .preventExtensions()
    .seal()
    .setPrototypeOf()
    .values()

Object.prototype
    .__defineGetter__()
    .__defineSetter__()
    __lookupGetter__()
    .__lookupSetter__()
    .hasOwnProperty()
    .isPrototypeOf()
    .propertyIsEnumerable()
    .toLocaleString()
    .toSource()
    .toString()
    .valueOf()
    .watch()
```

ENHANCED OBJECT PROPERTIES

```
let dest = { quux: 0 },
    src1 = { foo: 1, bar: 2 },
    src2 = { foo: 3, baz: 4 };

Object.assign(dest, src1, src2)

// dest.quux === 0;
// dest.foo  === 3;
// dest.bar  === 2;
// dest.baz  === 4
```

ECMAScript 5

```
var dest = { quux: 0 };
var src1 = { foo: 1, bar: 2 };
var src2 = { foo: 3, baz: 4 };

Object.keys(src1).forEach(function(k) { dest[k] = src1[k]; });
Object.keys(src2).forEach(function(k) { dest[k] = src2[k]; });

dest.quux === 0; dest.foo  === 3; dest.bar  === 2; dest.baz  === 4;
```

ENHANCED OBJECT PROPERTIES

```
let x = 0, y = 0;
let obj = { x, y };
```

```
ECMAScript 5
var x = 0, y = 0;
var obj = { x: x, y: y };
```

COMPUTED PROPERTY NAMES

```
let obj = {
    foo: "bar",
    ["baz" + quux()]: 42
}
```

```
ECMAScript 5
var obj = {
    foo: "bar"
};
obj[ "baz" + quux() ] = 42;
```

METHOD PROPERTIES

```
obj = {
     foo (a, b)  { ... },
     bar (x, y)  { ... },
    *quux (x, y) { ... }
}
```

```
ECMAScript 5
obj = {
    foo: function (a, b) {
        …
    },
    bar: function (x, y) {
        …
    },
    //  quux: no equivalent in ES5
    …
};
```

NEUE METHODEN VON STRING, NUMBER, MATH

STRING METHODS

```
String
     .fromCharCode()
     .fromCodePoint()
     .raw()

String.prototype
     .anchor()
     .big()
     .blink()
     .bold()
     .charAt()
     .charCodeAt()
     .codePointAt()
     .concat()
     .endsWith()
     .fixed()
     .fontcolor()
     .fontsize()
     .includes()
     .indexOf()
     .italics()
     .lastIndexOf()
     .link()
     .localeCompare()
     .match()
     .matchAll()
     .normalize()
     .padEnd()
     .padStart()
     .repeat()
     .replace()
     .search()
     .slice()
     .small()
     .split()
     .startsWith()
     .strike()
     .sub()
     .substr()
     .substring()
     .sup()
     .toLocaleLowerCase()
     .toLocaleUpperCase()
     .toLowerCase()
     .toSource()
     .toString()
     .toUpperCase()
     .trim()
     .trimRight()
     .trimLeft()
     .valueOf()
```

NEUE STRING METHODEN

```
         01234
let str='hello';

str.startsWith("ello", 1) // true  startet ab Position 1 mit ello
str.endsWith("hell",   4) // true  endet mit 'hell' mit Länge 4
str.includes("ell")       // true  enthält 'ello'
str.includes("ell", 1)    // true  enthält 'ell' ab Position 1
str.includes("ell", 2)    // false enthält 'ell' ab Position 2
```

```
ECMAScript 5
"hello".indexOf("ello")   === 1;                   // true
"hello".indexOf("hell")   === (4 - "hell".length); // true
"hello".indexOf("ell")    !== -1;                  // true
"hello".indexOf("ell", 1) !== -1;                  // true
"hello".indexOf("ell", 2) !== -1;                  // false
```

NUMBER TYPE CHECKING

```
Number.isNaN(42) === false
Number.isNaN(NaN) === true

Number.isFinite(Infinity) === false
Number.isFinite(-Infinity) === false
Number.isFinite(NaN) === false
Number.isFinite(123) === true
```

```
ECMAScript 5
var isNaN = function (n) {
    return n !== n;
};
var isFinite = function (v) {
    return (typeof v === "number" && !isNaN(v) && v !== Infinity && v !==
-Infinity);
};
```

NUMBER SAFETY CHECKING

```
Number.isSafeInteger(42) === true
Number.isSafeInteger(9007199254740992) === false
```

```
ECMAScript 5 – syntactic sugar: reduced | traditional
function isSafeInteger (n) {
    return (
           typeof n === 'number'
        && Math.round(n) === n
        && -(Math.pow(2, 53) - 1) <= n
        && n <= (Math.pow(2, 53) - 1)
    );
}
```

STANDARD EPSILON FÜR GENAUEREN FLIEßKOMMA VERGLEICH

```
0.1 + 0.2 === 0.3 // false

Math.abs((0.1 + 0.2) - 0.3) < Number.EPSILON // true
```

TRUNC

Ganzzahlermittlung:

```
Math.trunc(42.7)  // 42
Math.trunc( 0.1)  //  0
Math.trunc(-0.1)  // -0
```

ECMAScript 5

```
function mathTrunc (x) {
    return (x < 0 ? Math.ceil(x) : Math.floor(x));
}
```

NUMBER SIGN DETERMINATION

```
Vorzeichenbestimmung:

Math.sign(7)    //    1
Math.sign(0)    //    0
Math.sign(-0)   //   -0
Math.sign(-7)   //   -1
Math.sign(NaN)  // NaN
```

```
ECMAScript 5
function mathSign (x) {
    return (
        (x === 0 || isNaN(x)) ? x : (x > 0 ? 1 : -1)
    );
}
```

PROMISES

UNSERSTANDING PROMISES

- "Imagine you are a kid. Your mom promises you that she'll get you a new phone next week."
- You don't know if you will get that phone until next week. Your mom can either really buy you a brand new phone, or stand you up and withhold the phone if she is not happy.
- That is a promise.
- A promise has 3 states. They are:
- Pending: You don't know if you will get that phone
- Fulfilled: Mom is happy, she buys you a brand new phone
- Rejected: Your mom is happy, she withholds the phone

CREATING A PROMISE (ES6)

```
const isMomHappy = true;

// Promise
const willIGetNewPhone = new Promise(
    (resolve, reject) => {
        if (isMomHappy) {
            const phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone);
        } else {
            const reason = new Error('mom is not happy');
            reject(reason);
        }

    }
);
```

CONSUMING A PROMISE

```
// call our promise
const askMom = function () {
    willIGetNewPhone
        .then(showOff)
        .then(fulfilled => console.log(fulfilled))
        .catch(error => console.log(error.message));
};

askMom();
```

CHAINING PROMISES

- Promises are chainable.
- Let's say, you, the kid, promises your friend that you will show them the new phone when your mom buy you one.

CHAINING PROMISES

```
// 2nd promise

const showOff = function (phone) {
    const message = 'Hey friend, I have a new ' +
                phone.color + ' ' + phone.brand + ' phone';
    return Promise.resolve(message);
};
```

CALL YOUR PROMISES

```
var askMom = function () {
    willIGetNewPhone
    .then(showOff) // chain it here
    .then(function (fulfilled) {
            console.log(fulfilled);
         // output: 'Hey friend, I have a new black Samsung
phone.'
        })
        .catch(function (error) {
            // oops, mom don't buy it
            console.log(error.message);
         // output: 'mom is not happy'
        });
};
```

CREATING A PROMISE (ES7)

```
const isMomHappy = true;

// Promise
const willIGetNewPhone = new Promise(
    (resolve, reject) => {
        if (isMomHappy) {
            const phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone);
        } else {
            const reason = new Error('mom is not happy');
            reject(reason);
        }

    }
);
```

CHAINING PROMISES

```
// 2nd promise

async function showOff(phone) {
    return new Promise(
        (resolve, reject) => {
            var message = 'Hey friend, I have a new ' +
                phone.color + ' ' + phone.brand + ' phone';

            resolve(message);
        }
    );
};
```

CALL YOUR PROMISES

```
async function askMom() {
    try {
        console.log('before asking Mom');

        let phone = await willIGetNewPhone;
        let message = await showOff(phone);

        console.log(message);
        console.log('after asking mom');
    }
    catch (error) {
        console.log(error.message);
    }
}
```

https://scotch.io/tutorials/javascript-promises-for-dummies

GENERATOREN

GENERATOREN

- Generatorfunction und Generatorobject
- yield-Keyword und next-Methode

GENERATOR FUNCTION

```
function* range (start, end, step) {
    while (start < end) {
        yield start                                 // yield -> Ertrag
        start += step
    }
}

for (let i of range(0, 10, 2)) {
    console.log(i)                                  // 0, 2, 4, 6, 8
}
```

```
ECMAScript 5
function range (start, end, step) {
    var list = [];
    while (start < end) {
        list.push(start);
        start += step;
    }
    return list;
}

var r = range(0, 10, 2);
for (var i = 0; i < r.length; i++) {
    console.log(r[i]); // 0, 2, 4, 6, 8
}
```

GENERATOR, YIELD UND NEXT()

```
function* foo(index) {
  while (index < 2) {
    yield index++;
  }
}

const iterator = foo(0);

console.log(iterator.next().value);   // expected output: 0
console.log(iterator.next().value);   // expected output: 1
```

GENERATOR FUNCTION MIT ITERATOR

```
let fibonacci = {
  *[Symbol.iterator]() {
    let previous = 0, current = 1
    for (;;) {
      [ previous, current ] = [ current, previous + current ];
      yield current;
    }
  }
}
```

```
for (let n of fibonacci) {
    if (n > 1000)
        break
    console.log(n)
}
```

SYMBOLS

SYMBOL

```
const symbol1 = Symbol();
const symbol2 = Symbol(42);
const symbol3 = Symbol('foo');

console.log(typeof symbol1);
// expected output: "symbol"

console.log(symbol3.toString());
// expected output: "Symbol(foo)"

console.log(Symbol('foo') === Symbol('foo'));
// expected output: false
```

SYMBOL()

- The Symbol() function returns a value of type symbol
- Static properties that expose several members of built-in objects
- Static methods that expose the global symbol registry
- Resembles a built-in object class but is incomplete as a constructor because it does not support the syntax "new Symbol()"
- Every symbol value returned from Symbol() is unique.

SYMBOL

```
Properties

Symbol
    .asyncIterator
    .hasInstance
    .isConcatSpreadable
    .iterator
    .match
    .matchAll
    .prototype
    .prototype.description
    .replace
    .search
    .species
    .split
    .toPrimitive
    .toStringTag
    .unscopables
```

```
Methods

Symbol
    .for()
    .keyFor()

Symbol.prototype
    .toSource()
    .toString()
    .valueOf()
```

[SYMBOL.ITERATOR]

```
const iterable1 = new Object();

iterable1[Symbol.iterator] = function* () {
  yield 1;
  yield 2;
  yield 3;
};

console.log([...iterable1]); // expected output: Array [1, 2, 3]
```

ITERATOREN

FOR ... OF

```
let fibonacci = {
    [Symbol.iterator]() {
        let pre = 0, cur = 1
        return {
           next () {
               [ pre, cur ] = [ cur, pre + cur ]
               return { done: false, value: cur }
           }
        }
    }
}

for (let n of fibonacci) {
    if (n > 1000)
        break
    console.log(n)
}
```

SPREAD- UND RESTOPERATOR

- for-of-Schleife
- Bildung und Einsatz von Iteratoren

MAP/SET, WEAKMAP/WEAKSET

SET()

```
let s = new Set();
s.add("hello").add("goodbye").add("hello");

                                      // s.size === 2
                                      // s.has("hello") ===
true

for (let key of s.values()) {         // insertion order!!!
    console.log(key);
}
```

MAP()

```
let m = new Map();
let s = Symbol();

m.set("hello", 42);          // m.get(s) === 34
m.set(s, 34)                 // m.size === 2

for (let [ key, val ] of m.entries()) {
    console.log(key + " = " + val);
}
```

WEAKSET, WEAKMAP()

```
let isMarked = new WeakSet();

export class Node {
    constructor (id)   { this.id = id }
    mark        ()     { isMarked.add(this) }
    unmark      ()     { isMarked.delete(this) }
    marked      ()     { return isMarked.has(this) }
}

let foo = new Node("foo") // JSON.stringify(foo) === '{"id":"foo"}'

foo.mark()                // JSON.stringify(foo) === '{"id":"foo"}'
                          // isMarked.has(foo)     === true

foo = null                // remove only reference to foo
                          // isMarked.has(foo)     === false
```

WEAKSET, WEAKMAP()

```
let attachedData = new WeakMap()

export class Node {
    constructor (id)   { this.id = id                  }
    set data    (data) { attachedData.set(this, data)  }
    get data    ()     { return attachedData.get(this) }
}

let foo = new Node("foo")      // JSON.stringify(foo) === '{"id":"foo"}'

foo.data = "bar"               // foo.data === "bar"
                               // JSON.stringify(foo) === '{"id":"foo"}'
                               // attachedData.has(foo) === true

foo = null                     // remove only reference to foo
                               // attachedData.has(foo) === false
```

REFLECTIONS

REFLECT

```
let obj = { a: 1 }

Object.defineProperty(obj, "b", { value: 2 })
obj[Symbol("c")] = 3

console.log(Reflect.ownKeys(obj))        // [ "a", "b", Symbol(c) ]
```

REFLECT

Reflect.apply()
Ruft eine Zielfunktion mit Argumenten auf, die Argumente werden im Parameter args angegeben. Siehe auch Function.prototype.apply().

Reflect.construct()
Der new operator als Funktion. Equivalent zu new target(...args). Bietet die optionale Möglichkeit, einen anderen Prototyp anzugeben.

Reflect.defineProperty()
Ähnlich zu Object.defineProperty(). Gibt einen Boolean zurück.

Reflect.deleteProperty()
Der delete operator als Funktion. Ähnlich zu dem Aufruf delete target[name].

Reflect.get()
Eine Funktion, die den Wert von Eigenschaften/Properties zurückgibt.

Reflect.getOwnPropertyDescriptor()
Ähnlich zu Object.getOwnPropertyDescriptor(). Gibt einen Eigenschaftsdeskriptor der angegebenen Eigenschaft, oder undefined zurück.

Reflect.getPrototypeOf()
Gleich wie Object.getPrototypeOf().

REFLECT

Reflect.has()
Der in operator als Funktion. Gibt einen booleschen Wert zurück, der angibt, ob eine eigene oder geerbte Eigenschaft vorhanden ist.

Reflect.isExtensible()
Gleich wie Object.isExtensible().

Reflect.ownKeys()
Gibt ein Array der eigenen (nicht geerbten) Eigenschaftsschlüssel des Zielobjekts zurück.

Reflect.preventExtensions()
Ähnlich zu Object.preventExtensions(). Gibt einen Boolean zurück.

Reflect.set()
Eine Funktion, die den Eigenschaften/Properties Werte zuweist. Gibt einen Booleanzurück, der true ist, wenn die Zuweisung erfolgreich verlief.

Reflect.setPrototypeOf()
Eine Funktion, die den Prototyp eines Objekts festlegt.