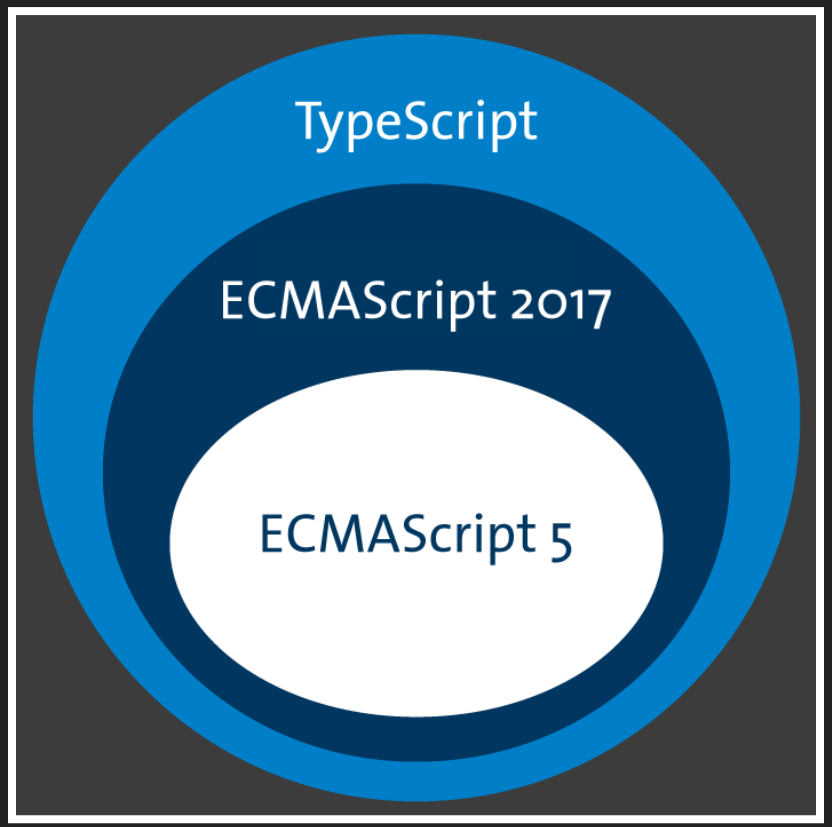
HERR OBER: EINE GETYPTE OBERMENGE VON JAVASCRIPT BITTE

JOHANNES DIENST

TYPESCRIPT

JAVASCRIPT FÜR BACKENDENTWICKLER


TypeScript
ECMAScript 2017
ECMAScript 5

```
interface Robot {
    model :string;
    canSpeak() :boolean;
}
```

```
class Terminator implements Robot {

    constructor(public model :string){}

    canSpeak() :boolean {
        return true;
    }
}
```

```
function sugar(p1, p2="42", p3?) {
  console.log(p1); // Hello
  console.log(p2); // 42
  console.log(p3); // undefined
}

sugar('Hello');
```

INSTALLATION

```
npm install -g typescript

tsc helloworld.ts
```

```
{
  "devDependencies": {
    "typescript": "^1.8.10"
  },
  "scripts": {
    "watch:tsc": "./node_modules/.bin/tsc -w",
    "watch": "npm run watch:tsc",
    "start": "node src/js/livecoding.js"
  },
  "dependencies": {
    "request": "*"
  }
}
```

```
{
    "compilerOptions": {
        "module": "commonjs",
        "target": "ES6",
        "noImplicitAny": false,
        "rootDir": "./src/ts",
        "outDir": "./src/js",
        "sourceMap": false,
        "removeComments": true
    },
    "exclude": [
        "node_modules"
    ]
}
```

```
{
  "rules": {
    "variable-name": [
      true,
      "check-format",
      "allow-leading-underscore",
      "ban-keywords"
    ]
  }
}
```

BASISTYPEN

```
let isDone :boolean = false;

let decimal :number = 6;

let color :string = "berlin";
```

```
let aArray :number[ ] = [1, 2, 3];

enum Color {Red, Green, Blue};

let c: Color = Color.Green;
```

```
let notSure :any = 4;
```

OBJEKTORIENTIERUNG

Interfaces

Klassen

Datenkapselung

INTERFACES

```
interface Robot {
    name :string;
    canSpeak() :boolean;
}

function printName(aRobot :Robot) {
    console.log(aRobot.name);
}

let T800 =
    {name: 'Arnold', canSpeak: function(){return false;}};

printName(T800);
```

```
interface Animal {
    name? :string;
    color? :string;
}
```

```
interface Human extends Animal {
    hugeBrain :boolean;
}

let aHuman :Human = {name: 'Homo sapiens', hugeBrain: true};
```

KLASSEN

```
class Dog implements Animal {
    name :string;

    constructor(name :string, public color :string) {}

    getColor() {
        return this.color;
    }
}
```

DATENKAPSELUNG

```
class Labrador {
    firstname :string;
    public lastname :string;
    protected age :number;
    private gender :string;

    constructor(firstname :string, lastname :string,
      age :number, gender :string) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.age = age;
        this.gender = gender
    }
}
```

```
class GoldenRetriever extends Labrador {

    public getFirstname() {
        return this.firstname;
    }

    public getLastname() {
        return this.lastname;
    }

    public getAge() {
       return this.age;
    }
}
```

```
// Private property!
public getGender() {
    return this.gender;
}
```

FUNKTIONEN

FUNKTIONSTYPEN

```
function printDogName(aDog :Dog) {
    console.log(aDog.name);
}

let printFunc :(Dog) => void;
printFunc = printName;

// Error
let printFunc2 :(Dog) => boolean;
printFunc2 = printName;
```

DEFAULT/OPTIONALE PARAMETER

```
function funcDefault(p1="World", p2? :number) {
    console.log('Hello ' + p1 + ' ' + p2);
}

// Hello Buenos dias  42
funcDefault("Buenos dias ", 42);

// Hello World undefined
funcDefault();
```

REST-PARAMETER

```
// Rest parameter
function buildName(firstName :string,
    ...restOfName :string[]) {
      return firstName + " " +
             restOfName.join(" ");
}
```

GENERICS

```
// Generic function
function identity<T>(arg :T) :T {
    return arg;
}


function loggingIdentity<T>(arg :T) :T {
    // Error: T doesn't have .length
    console.log(arg.length);
    return arg;
}
```

```
// Generic constraint
interface Lengthwise {
    length :number;
}

function loggingIdentity<T extends Lengthwise> (arg :T) :T {
    console.log(arg.length);  // No error
    return arg;
}
```

```
// Generic class
class GenericNumber<T> {
    zeroValue :T;
    add :(x :T, y :T) => T;
}

let myGenericNumber =
    new GenericNumber<number>();

myGenericNumber.zeroValue = 0;

myGenericNumber.add =
    function(x, y) { return x + y; };
```

MIXINS

```
function applyMixins(derivedCtor :any, baseCtors :any[]) {
    baseCtors.forEach(baseCtor => {
        Object.getOwnPropertyNames(baseCtor.prototype).
            forEach(name => {
                derivedCtor.prototype[name] =
                    baseCtor.prototype[name];
        });
    });
}
```

```
class Person {
    name :string;
}

class ConsoleLogger {
    log() {}
}
```

```
class PersonLogger implements Person, ConsoleLogger {
    constructor(public name :string){}
    log :()=>void;
}
```

```
applyMixins(
    PersonLogger,
    [new Person("Jim"), new ConsoleLogger()]);

var jim = new PersonLogger("Jim");
var n = jim.name;
jim.log();
```

UNION TYPES

```
interface C3PO {
    move();
    talk();
}

interface R2D2 {
    move();
    whistle();
}

function getRobot() :C3PO | R2D2 {
    // ...
}
```

```
let robot = getRobot();

robot.move(); // okay

robot.talk(); // error
```

TYPE ALIASES

```
type Name = string;

type NameResolver = () => string;

type NameOrResolver = Name | NameResolver;
```

```
function getName(n :NameOrResolver): Name {
    if (typeof n === 'string') {
        return n;
    }
    else
    {
        return n();
    }
}
```

STRING LITERAL TYPES

```
type Easing = "ease-in" | "ease-out" |
    "ease-in-out";
```

```
class UIElement {
    animate(dx :number, dy :number,
        easing :Easing) {
        ...
    }
}
```

```
let button = new UIElement();

button.animate(0, 0, "ease-in");

button.animate(0, 0, "uneasy"); // error
```

POLYMORPHIC THIS TYPES

```
class BasicCalculator {

    public constructor(
        protected value :number = 0) { }

    public currentValue() :number {
        return this.value;
    }

    public multiply(operand: number) :this {
        this.value *= operand;
        return this;
    }
}
```

```
let v = new BasicCalculator(2)
            .multiply(5)
            .currentValue();
```

HYBRID TYPES

```
interface Counter {
    (start: number) :string;
    interval :number;
    reset() :void;
}
```

```
function getCounter() :Counter {
    let counter = <Counter>function (start :number) { };
    counter.interval = 123;
    counter.reset = function () { };
    return counter;
}
```

```
let count :Counter = getCounter();
count(10);
count.reset();
count.interval = 5.0;
```

INTERFACES ERWEITERN KLASSEN

```
class Control {
    private state :any;
}

interface SelectableControl extends Control {
    select() :void;
}
```

```
class Button extends Control {
    select() { }
}

class Image extends Control {
}

class Location {
    select() { }
}
```

INDEX TYPEN

```
interface StringArray {
    [index :number] :string;
}

let myArray :StringArray;
myArray = ["Bob", "Fred"];

let myStr :string = myArray[0];
```

JohannesDienst

johannesdienst.net

jdienst@multamedio.de