

ECMAScript - Javascript

Historie Namensgebung

Ecmascript wird seit etwa 1997 als Standard für Javascript und ähnliche Sprachen entwickelt. Die Versionen wurden einfach aufwärts nummeriert bis Ecmascript 6 welches dann nachträglich in Ecmascript2015 umbenannt wurde. Die aktuelle Bezeichnung für Ecmascript ist die mit Jahreszahl aber in der Praxis sind durchaus auch die alten Namen gebräuchlich, welche sogar noch weiter gezählt werden wie Ecmascript 9 oder als Kurzform ES 9 beweist.

Quelle: <https://en.wikipedia.org/wiki/ECMAScript>

Browserkompatibilität und Transpiler

Da es immer eine längere Zeit dauert bis die neuen Features einer neuen Ecmascript Version in allen Browsern umgesetzt ist, werden Transpiler benutzt um von der neuen Ecmascript Version auf die aktuell überall implementierte Version zu übersetzen. Transpiler sind uns aus der Compilertechnik noch als Querübersetzer in Erinnerung welche eine Hochsprache in eine andere Hochsprache übersetzt haben, z.B. Oberon nach C damit Oberon auch für weitere Betriebssysteme bereitgestellt werden konnte (fiktives Beispiel für Oberon). Die aktuelle Version wird dann vom echten Compiler (in Node.js oder Chrome Browser z.B. V8) in nativen OS Kode übersetzt..

Quellen:

* <https://www.ab-heute-programmieren.de/es2015-teil-1-was-ist-es2015/>

* <https://en.wikipedia.org/wiki/ECMAScript>

ES6 - Feature: Spread-Syntax

Mit Hilfe der Spread Syntax lassen sich anstelle von kommagetrennten Argumentlisten einfach Teile eines oder auch ganze Arrays übergeben.

Beispiele

```
function myFunction(v, w, x, y, z) { }
var args = [0, 1];
myFunction(-1, ...args, 2, ...[3]);
var parts = ['shoulders', 'knees'];
var lyrics = ['head', ...parts, 'and', 'toes']; //
["head", "shoulders", "knees", "and", "toes"]
```

Quelle: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Operators/Spread_operator

ES6 - Feature: Klassen

Was sind Klassen

Das Schlüsselwort class gibt es seit ES6 (Ecma - script Version 6). Es handelt sich um syntaktischen Zucker durch den die class Syntax auf eine Funktion abgebildet wird. Also sind Klassen eigentlich nichts weiter als Funktionen aber schöner lesbar.

Anbei der Beweis:

```
'use strict';

class Foo{}

console.log(typeof Foo);

> function

Der Typ einer Klasse ist also eine Funktion und nicht wie erwartet eine Metaklasse.

// Klassen definieren

class Person {

  constructor( firstName, lastName) {

    this.firstName = firstName;
    this.lastName = lastName;

  }

  greet() {

    console.log('Hallo ' + this.firstName + ' ' + this.lastName);

  }

}

// Vererbung von Klassen

var john = new Person('John', 'Maier');
john.greet();

class Child extends Person {

  constructor( firstName, lastName, age) {

    super( firstName, lastName);
    this.age = age;

  }

  greet() {

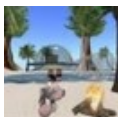
    super.greet();

    console.log('Du bist ' + this.age + ' Jahre alt!');

  }

}

// Statische Methoden
```



By **Huluvu424242**
(FunThomas424242)

Published 14th July, 2019.
Last updated 20th July, 2019.
Page 1 of 2.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

ES6 - Feature: Klassen (cont)

```
> class Child extends Person {
  static isTeenager(child) {
    if (child.age >= 10 && child.age <= 19) {
      return true;
    }
    return false;
  }
}
```

Achtung: Bei Klassen gibt es kein hoisting! Daraus folgt, die Benutzung einer nicht vorher definierten Klasse führt zum ReferenceError.

Quellen:

* <https://www.techlounge.io/node#episode-204647256> Folge 7:

Callbacks verwenden

* <https://www.ab-heute-programmieren.de/es2015-teil-3-klassen/>
(Beispiele komplett übernommen)

ES6 - Feature: Pfeilfunktionen

Alte Syntax

```
doSomething( function( result ) {
  console.log('Das Ergebnis ist: ' + result);
});
```

Neue Syntax

```
doSomething( result => console.log('Das Ergebnis ist: ' + result));
```

Achtung: Semantischer Unterschied ist this! Während this sich bei der alten Syntax nur auf den Scope innerhalb der Funktion bezieht, bezieht sich this bei der neuen Syntax auf Dinge aus der Aufrufenden Umgebung der Funktion.

Quelle: <https://www.ab-heute-programmieren.de/es2015-teil-2-pfeil-funktionen/>

Beweis mit Plunker:

```
var test = {
  a : 10,
  b : [1, 2, 3],
  add : function() {
    this.b.forEach( function( b ) {
      console.log( this.a + b );
    });
  }
}
```

ES6 - Feature: Pfeilfunktionen (cont)

```
> test.add();
>NaN
>NaN
>NaN
var test = {
  a : 10,
  b : [1, 2, 3],
  add : function() {
    this.b.forEach(b => console.log(this.a + b));
  }
}
```

```
test.add();
```

```
>11
```

```
>12
```

```
>13
```

Quelle: <https://www.ab-heute-programmieren.de/es2015-teil-2-pfeil-funktionen/>

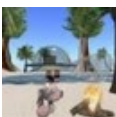
(Beispiel komplett übernommen)

asynch / await

Es gelten folgende Regeln:

- * await gibt immer ein aufgelöstes Promise zurück oder es wirft einen unbehandelten Fehler
 - * await verpackt simple Werte ebenfalls in ein aufgelöstes Promise und gibt dieses zurück
 - * await wartet bis eine nachfolgende Promise aufgelöst wurde
 - * await ist nur innerhalb einer mit asynch markierten Funktion möglich
 - * asynch kennzeichnet eine asynchrone Funktion
- Eine Alternative zu await ist die Behandlung der Promise mittels then und catch. Nach Anwendung dieser Alternativen, wird keine Markierung der umschließenden Funktion mit asynch mehr notwendig.

Quelle: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Statements/async_function



By **Huluvu424242**
(FunThomas424242)

Published 14th July, 2019.
Last updated 20th July, 2019.
Page 2 of 2.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>