

Javascript - Currying

Was ist das?

Beim Currying wird eine Funktion der Ordnung n+1 (also n+1 Parameter) auf eine Funktion der Ordnung n (also n Parameter) reduziert.

Das Konzept dabei ist, die Funktion um einen Parameter zu kürzen und statt des ursprünglichen, berechneten Wertes eine Funktion mit dem gekürzten Parameter zur weiteren Berechnung zurück zu geben. Damit wird die Funktion nun mit einem Parameter weniger aufgerufen. Nun muss aber das Ergebnis noch mit dem reduzierten Parameter aufgerufen werden. Also in Summe wurden wieder alle Parameter zur Berechnung übergeben, nur verteilt auf 2 Aufrufe oder mehr.

Beispiel für Currying und Uncurrying

```
curryMultiply = (n) => (m) => n * m
curryMultiply(3)(4) === 12 // true
multiply = (n, m) => curryMultiply(n)(m)
multiply(3, 4) === 12 // true
```

Quelle: <https://blog.benestudio.co/currying-in-javascript-es6-540d2ad09400>

Javascript - Ich finde Lambdas verkürzt doof

Javascript - Ich finde Lambdas verkürzt doof (cont)

```
// vollständige Schreibweise mit Selbstaufführung
am Ende
const berechneWortlaenge = function(){
  const map = processArray(woerter, function( wort
){
  return { wort: wort, laenge: wort.length};
});
  console.log('1. Ausgabe: ' +JSON.stringify(-
(map));
}());
// Umgewandelt in lambda Ausdruck
const berechneWortlaenge = function(){
  const map = processArray(woerter, ( wort ) =>{
    return { wort: wort, laenge: wort.length};
  });
  console.log('2. Ausgabe: ' + JSON.stringify(-
map));
}();
// obsolete Klammern entfernt: Parameterliste,
Body und
// return Schlüsselwort entfernt, da an letzter
Stelle
// => compile Fehler weil das Ergebnisobjekt als
Body
// interpretiert wird !!!
//const berechneWortlaenge = function(){
// const map = processArray(woerter, wort =>
// { wort: wort, laenge: wort.length}
// );
// console.log('3. Ausgabe: ' + JSON.stringify(-
map));
//}();
// Obsolete Klammern zur Umwandlung in Expression
hinzugefügt
const berechneWortlaenge = function(){
  const map = processArray(woerter, wort =>
    ({ wort: wort, laenge: wort.length})
  );
  console.log('4. Ausgabe: ' + JSON.stringify(-
map));
}();
```

```
// Lambdas sind so eine schöne Sache aber diese
ständige Verkürzung
// soweit wie geht ist einfach nervig und birgt
Fehler.
// Zwei Klammern für die Parameterliste und zwei
für den Body kann man
// doch schreiben oder?
// Sie sind anderer Meinung? - is ok.
// Ich nehme an Sie wissen, dass Pfeilfunktionen
rechts assoziativ sind.
// Trotzdem glaube ich nicht, dass Sie das hier
lesen wollen:
curry = f => a => b => f(a, b)
uncurry = f => (a, b) => f(a)(b)
papply = (f, a) => b => f(a, b)
Quelle: https://blog.benestudio.co/currying-in--
javascript-es6-540d2ad09400
// Anderes und hoffentlich besseres Beispiel
// Nur zur Vorbereitung
const woerter = ['Hallo', 'heute', 'gestern'];
const processArray = function( items, fn){
  const result = [];
  items.forEach( (item) => result.push(fn(item)));
  return result;
}
```



By **Huluhu424242**
(FunThomas424242)

Published 28th April, 2019.
Last updated 2nd July, 2019.
Page 1 of 4.

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

cheatography.com/funthomas424242/
stackoverflow.com/users/story/373498

Javascript - Operatoren

a + b	Addition von a und b
a - b	Subtraktion: subtrahiert b von a
a * b	Multiplikation von a und b
x ** y	Exponentialfunktion: x hoch y (wobei sowohl x als auch y variabel sein können - leider keinen korrekten mathematischen Namen gefunden: Potenzfunktion wäre fester Exponent, also auch nicht korrekt.)
a / b	Division: a durch b
a % b	Modulus: a modulo b
a++	Inkrement: erhöht a um 1
a--	Dekrement: verringert a um 1
==	Gleichheitsprüfung nur auf Wertebasis (verschiedene aber wertgleiche Datentypen ergeben true)
===	Gleichheitsprüfung inklusive Werte- und Typgleichheit
!=	Ungleich ohne Typberücksichtigung
!==	Ungleich mit Typberücksichtigung
>	größer als
<	kleiner als
>=	größer gleich
<=	kleiner gleich
?	Ternärer Operator: (bedingung) ? ifAnweisung : elseAnweisung;
&&	logisches UND
	logisches ODER
!	logisches NOT
typeof <a>	Bestimmt den Typ der Variablen a
o instanceof t	true falls o ein Objekt vom Typ t ist
&	bitweises UND
	bitweises ODER
~	bitweises NOT
^	bitweises XOR

Javascript - Operatoren (cont)

<<	bitweises links Schieben; mit 0 auffüllend
>>	bitweises rechts Schieben; vorzeichenerhaltend (so die Theorie)
>>>	bitweises rechts Schieben; mit 0 auffüllend

Quelle: https://www.w3schools.com/js/js_operators.asp

Javascript - Datentypen

In Javascript existieren 5 primitive Datentypen:

- * String = Zeichenkette
- * Number = Ziffernfolge
- * Boolean = Boolescher Datentyp
- * Null = leere Referenz
- * Undefined = Nicht vorhandener Wert

Quelle: <https://www.ab-heute-programmieren.de/js-teil-3-datentypen-und-hoisting/>

Javascript Konzept: Hoisting

Beim Hoisting werden später definierte Variablen bereits am Scopebeginn (für den Entwickler unsichtbar) automatisch deklariert. Dadurch können Wertzuweisungen auf eine Variable erfolgen, welche laut Quellcode noch gar nicht existiert.

**Beispiel*

```
num = 6;
console.log(num); // Gibt 6 zurück
var num = 7;
console.log(num); // Gibt 7 zurück
Hoisting gibt es auch für Funktionen
```

Beispiel

```
catName("Chloe");
function catName(name) {
    console.log("Der Name meiner Katze ist " +
name);
}
```

*/**

Das Ergebnis des Codes oben ist: "Der Name meiner Katze ist Chloe"

**/*



By **Huluvu424242**
(FunThomas424242)

Published 28th April, 2019.
Last updated 2nd July, 2019.
Page 2 of 4.

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

Javascript Konzept: Hoisting (cont)

Wichtig: Es werden nur die Deklarationen gehoisted, nicht die Zuweisungen.

Beispiele

```
console.log(num); // Gibt undefined zurück
var num;
num = 6;
var x = 1; // Initialisiere x
console.log(x + " " + y); // '1 undefined'
var y = 2; // Initialisiere y
// Das obige Beispiel wird implizit als das
folgende verstanden:
var x; // Deklariere x
var y; // Deklariere y
// Hoisting beendet.
x = 1; // Initialisiere x
console.log(x + " " + y); // '1 undefined'
y = 2; // Initialisiere y
```

Quelle: <https://developer.mozilla.org/de/docs/Glossary/Hoisting>

Javascript - Typische Datentypkonvertierungen

25 + " Zahl -> String, Die Zahl 25 wird zu '25'

'25' - 0 String -> Zahl, Der Text '25' wird zur Zahl 25

!!a any -> boolean, Der Wert der Variablen a wird zum Wahrheitswert: true falls a einen Wert enthält und false falls nicht

Umwandlung einer Promise Callback Hölle

Alter Stil:

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult)
    {
      console.log('Got the final result: ' +
finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Moderner Stil:

```
doSomething().then(function(result) {
  return doSomethingElse(result);
```

Umwandlung einer Promise Callback Hölle (cont)

```
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalR-
esult);
})
.catch(failureCallback);
```

Der alte Stil kann in den neuen Stil überführt werden, wenn es nur darum geht, dass die inneren Promises nach den äußeren ausgeführt werden und die äußeren Variablen in den inneren Promises keine Rolle mehr spielen.

https://developer.mozilla.org/de/docs/Web/JavaScript/Guide/Using_promises

Overloading von Funktionssignaturen

```
export class Test{

  public add( a: Number, b:Number):void;
  public add( a: String, b:String):void;
  public add( a: boolean, b:boolean):void;
  public add( a, b):void{
    console.log(a+b);
  }

  public static run():any{
    let t:Test = new Test();
    t.add('a','b');
    t.add(3,5);
    t.add(true,true);
  }
}

Test.run();

Erzeugt die Ausgaben:
ab
8
2
```



By **Huluvu424242**
(FunThomas424242)

Published 28th April, 2019.
Last updated 2nd July, 2019.
Page 3 of 4.

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