3

Funktionen und funktionale Programmierung

In diesem Kapitel erfahren Sie, wie Sie in JavaScript Funktionen schreiben. JavaScript ist eine funktionale Programmiersprache. Funktionen sind ebenso wie Zahlen oder Strings Werte erster Klasse. Sie können Funktionen entgegennehmen und andere Funktionen ausgeben. Für die Arbeit mit modernem JavaScript ist es unverzichtbar, die Kunst der funktionalen Programmierung beherrschen zu lernen.

Des Weiteren behandelt dieses Kapitel die Regeln für die Parameterübergabe und den Gültigkeitsbereich in JavaScript sowie das Auslösen und Abfangen von Exception.

3.1 Funktionen deklarieren

Um in JavaScript eine Funktion zu deklarieren, müssen Sie Folgendes angeben:

1. den Namen der Funktion,
2. die Namen der Parameter,
3. den Rumpf der Funktion, der das Funktionsergebnis berechnet und zurückgibt.

Die Typen der Funktionsparameter und des Ergebnisses geben Sie dagegen nicht an. Betrachten Sie dazu das folgende Beispiel:

```
function average(x, y) {
  return (x + y) / 2
}
```

Die Anweisung `return` sorgt dafür, dass der in der Funktion berechnete Wert zurückgegeben wird.

Um diese Funktion aufzurufen, müssen Sie lediglich die erforderlichen Argumente übergeben:

```
let result = average(6, 7) // result wird auf 6.5 gesetzt
```

Was aber geschieht, wenn Sie etwas anderes als Zahlen übergeben? Was auch immer passiert, passiert. Zum Beispiel:

```
result = average('6', '7') // result wird auf 33.5 gesetzt
```

Wenn Sie Strings übergeben, werden sie durch den Operator + im Funktionsrumpf verkettet. Der resultierende String `'67'` wird dann vor der Division durch 2 in eine Zahl umgewandelt.

Auf Java-, C#- oder C++-Programmierer, die an eine Typüberprüfung zur Kompilierzeit gewöhnt sind, mag das geradezu fahrlässig wirken. Tatsächlich können Sie Fehler bei den Argumenttypen erst erkennen, wenn zur Laufzeit merkwürdige Dinge passieren. Aber dadurch ist es möglich, Funktionen zu schreiben, die Argumente verschiedener Typen akzeptieren, was sehr komfortabel sein kann.

Die Anweisung `return` gibt die Steuerung unmittelbar zurück und verwirft den Rest der Funktion. Betrachten Sie dazu die folgende Funktion `indexOf`, die den Index eines Wertes in einem Array berechnet:

```
function indexOf(arr, value) {
  for (let i in arr) {
    if (arr[i] === value) return i
  }
 return -1
}
```

Sobald eine Übereinstimmung gefunden wurde, wird der Index zurückgegeben und die Funktion beendet.

In einer Funktion muss nicht unbedingt ein Rückgabewert angegeben sein. Wenn der Funktionsrumpf ohne `return` beendet wird oder wenn kein Ausdruck auf das Schlüsselwort `return` folgt, gibt die Funktion `undefined` zurück. Das geschieht gewöhnlich, wenn die Funktion einzig und allein aufgrund ihrer Seiteneffekte aufgerufen wird.

Tipp

Wenn eine Funktion nur manchmal ein Ergebnis zurückgibt, Sie aber nicht wollen, dass Sie jemals irgendetwas zurückgibt, dann legen Sie das explizit fest:

```
return undefined
```

Hinweis

Wie in Kapitel 2 bereits erwähnt, muss bei einer `return`-Anweisung immer mindestens ein Token vor dem Zeilenende stehen, um zu verhindern, dass automatisch ein Semikolon eingefügt wird. Wenn beispielsweise eine Funktion ein Objekt zurückgibt, müssen Sie daher wenigstens die öffnende geschweifte Klammer in dieselbe Zeile schreiben:

```
return {
  average: (x + y) / 2,
  max: Math.max(x, y),
  ...
}
```

3.2 Funktionen höherer Ordnung

JavaScript ist eine funktionale Programmiersprache. Funktionen sind Werte und können daher in Variablen gespeichert, als Argumente übergeben und als Ergebnisse von anderen Funktionen zurückgegeben werden.

Beispielsweise können wir die Funktion `average` in einer Variablen speichern:

```
let f = average
```

Anschließend können Sie die Funktion wie folgt aufrufen:

```
let result = f(6, 7)
```