Parameter und Rückgabewerte

Werte an Funktionen übergeben

Bisher können Funktionen feste Aufgaben ausführen. Häufig sollen sie aber mit *unterschiedlichen Daten* arbeiten. Dafür gibt es **Parameter**.

Ein Parameter ist ein Platzhalter in der Funktionsdefinition. Beim Aufruf erhält dieser Platzhalter einen konkreten Wert. Dieser konkrete Wert heißt **Argument**.

```
function begrüße(name) {
  console.log("Hallo, " + name + "!");
}

begrüße("Mira");
begrüße("Jonas");
```

Die Ausgabe lautet:

```
Hallo, Mira!
Hallo, Jonas!
```

`name` ist der Parameter. `"Mira"` und `"Jonas"` sind die Werte, die beim jeweiligen Funktionsaufruf übergeben werden.

Parameter machen Funktionen wiederverwendbar

Ohne Parameter müsste für jede Begrüßung eine eigene Funktion geschrieben werden:

```
function begrüßeMira() {
  console.log("Hallo, Mira!");
}

function begrüßeJonas() {
  console.log("Hallo, Jonas!");
}
```

Das funktioniert, ist aber unpraktisch. Mit einem Parameter reicht eine einzige Funktion:

```
function begrüße(name) {
  console.log("Hallo, " + name + "!");
}
```

Diese Funktion kann nun für beliebig viele Namen verwendet werden.

```
begrüße("Aylin");
begrüße("Ben");
begrüße("Chris");
```

Mehrere Parameter verwenden

Eine Funktion kann auch mehrere Parameter haben. Sie werden durch Kommas getrennt.

```
function addiere(zahl1, zahl2) {
  console.log(zahl1 + zahl2);
}

addiere(4, 7);
```

Die Ausgabe ist:

```
11
```

JavaScript ordnet die Argumente der Reihenfolge nach zu:

```
addiere(4, 7);
```

- `zahl1` erhält den Wert `4`.

- `zahl2` erhält den Wert `7`.

Ein weiteres Beispiel berechnet den Preis mehrerer gleicher Artikel:

```
function berechneGesamtpreis(preis, anzahl) {
  console.log(preis * anzahl);
}

berechneGesamtpreis(3.5, 4);
```

Die Funktion gibt in der Konsole `14` aus.

> **Wichtig:** Die Namen der Parameter kannst du frei wählen. Sie sollten aber klar beschreiben, welche Werte erwartet werden.

Gut lesbar:

```
function berechneGesamtpreis(preis, anzahl) {
  console.log(preis * anzahl);
}
```

Weniger verständlich:

```
function berechneGesamtpreis(a, b) {
  console.log(a * b);
}
```

Parameter sind lokale Variablen

Innerhalb einer Funktion verhalten sich Parameter wie Variablen. Du kannst mit ihnen rechnen, sie vergleichen oder in Texten verwenden.

```
function prüfeAlter(alter) {
  if (alter >= 18) {
    console.log("Du bist volljährig.");
  } else {
    console.log("Du bist noch nicht volljährig.");
```

```
  }
}

prüfeAlter(20);
prüfeAlter(15);
```

Die Ausgabe:

```
Du bist volljährig.
Du bist noch nicht volljährig.
```

Der Parameter `alter` existiert nur innerhalb der Funktion `prüfeAlter`.

```
function zeigeZahl(zahl) {
  console.log(zahl);
}

zeigeZahl(8);

console.log(zahl);
```

Die letzte Zeile führt zu einem Fehler, weil `zahl` außerhalb der Funktion nicht bekannt ist.

Standardwerte für Parameter

Manchmal wird beim Aufruf kein Wert übergeben. Dann kann ein **Standardwert** hilfreich sein.

```
function begrüße(name = "Gast") {
  console.log("Hallo, " + name + "!");
}

begrüße("Lea");
begrüße();
```

Die Ausgabe lautet:

```
Hallo, Lea!
Hallo, Gast!
```

Wird kein Argument angegeben, verwendet JavaScript den Standardwert `"Gast"`.

Auch mehrere Parameter können Standardwerte besitzen:

```
function zeigeNachricht(text = "Keine Nachricht", dauer = 3) {
  console.log(text + " – Anzeige für " + dauer + " Sekunden");
}

zeigeNachricht("Gespeichert", 5);
zeigeNachricht();
```

Werte mit `return` zurückgeben

Eine Funktion kann nicht nur etwas in der Konsole ausgeben. Sie kann auch ein Ergebnis an die Stelle zurückgeben, an der sie aufgerufen wurde. Dafür verwendest du `return`.

```
function addiere(zahl1, zahl2) {
  return zahl1 + zahl2;
}

let ergebnis = addiere(4, 7);

console.log(ergebnis);
```

Die Ausgabe ist:

```
11
```

Hier passiert Folgendes:

1. Die Funktion `addiere(4, 7)` wird aufgerufen.
2. Sie berechnet `4 + 7`.
3. `return` gibt das Ergebnis `11` zurück.
4. Das Ergebnis wird in der Variablen `ergebnis` gespeichert.
5. `console.log(ergebnis)` gibt den gespeicherten Wert aus.

Du kannst die Funktion auch direkt in `console.log()` verwenden:

```
console.log(addiere(10, 5));
```

Oder ihr Ergebnis für weitere Berechnungen nutzen:

```
let summe = addiere(8, 2);
let verdoppelteSumme = summe * 2;

console.log(verdoppelteSumme);
```

Die Ausgabe lautet:

```
20
```

Der Unterschied zwischen `console.log()` und `return`

Diese beiden Anweisungen werden leicht verwechselt, haben aber unterschiedliche Aufgaben.

```
function zeigeSumme(zahl1, zahl2) {
  console.log(zahl1 + zahl2);
}
```

Diese Funktion zeigt das Ergebnis nur in der Konsole an. Das Ergebnis kann danach nicht weiterverwendet werden.

```
let ergebnis = zeigeSumme(2, 3);

console.log(ergebnis);
```

Die Konsole zeigt:

```
5
undefined
```

Warum erscheint `undefined`? Die Funktion hat keinen Wert zurückgegeben. Ohne `return` endet eine Funktion automatisch mit dem Rückgabewert `undefined`.

Mit `return` funktioniert es:

```
function berechneSumme(zahl1, zahl2) {
  return zahl1 + zahl2;
}

let ergebnis = berechneSumme(2, 3);

console.log(ergebnis);
```

Die Ausgabe:

```
5
```

Merke:

- `console.log()` zeigt einen Wert zur Kontrolle in der Konsole.
- `return` gibt einen Wert aus einer Funktion zurück.
- Einen Rückgabewert kannst du speichern, weiterrechnen oder an eine andere Funktion übergeben.

`return` beendet die Funktion

Sobald JavaScript auf `return` trifft, wird die Funktion sofort beendet. Code darunter wird nicht mehr ausgeführt.

```
function prüfeZahl(zahl) {
  if (zahl < 0) {
    return "Die Zahl ist negativ.";
  }

  console.log("Diese Ausgabe erfolgt nur bei null oder positiven Zahlen.");
  return "Die Zahl ist null oder positiv.";
}

console.log(prüfeZahl(-3));
console.log(prüfeZahl(5));
```

Für `-3` wird die `console.log()`-Zeile innerhalb der Funktion übersprungen. Die Funktion endet direkt bei `return`.

Dieses Verhalten ist besonders nützlich, um Fälle früh zu behandeln:

```
function teile(dividend, divisor) {
  if (divisor === 0) {
    return "Division durch null ist nicht möglich.";
  }

  return dividend / divisor;
}

console.log(teile(12, 3));
console.log(teile(12, 0));
```

Die Ausgabe:

```
4
Division durch null ist nicht möglich.
```

Rückgabewerte mit Bedingungen

Funktionen können abhängig von einer Bedingung unterschiedliche Werte zurückgeben.

```
function istVolljährig(alter) {
  if (alter >= 18) {
    return true;
  } else {
    return false;
  }
}

console.log(istVolljährig(21));
console.log(istVolljährig(16));
```

Die Ausgabe:

```
true
false
```

Da der Vergleich `alter >= 18` selbst bereits `true` oder `false` liefert, kann die Funktion noch kürzer geschrieben werden:

```
function istVolljährig(alter) {
  return alter >= 18;
}
```

Du kannst diese Funktion direkt in einer Bedingung einsetzen:

```
if (istVolljährig(19)) {
  console.log("Zugang erlaubt.");
} else {
  console.log("Zugang nicht erlaubt.");
}
```

Ein praktisches Beispiel: Rabatt berechnen

Die folgende Funktion berechnet einen neuen Preis nach einem prozentualen Rabatt.

```
function berechneRabattPreis(preis, rabattProzent) {
  let rabatt = preis * rabattProzent / 100;
  return preis - rabatt;
}

let neuerPreis = berechneRabattPreis(80, 25);

console.log(neuerPreis);
```

Die Ausgabe lautet:

```
60
```

Die Funktion erhält zwei Werte:

- `preis`: der ursprüngliche Preis
- `rabattProzent`: der Rabatt in Prozent

Sie berechnet zunächst den Rabatt und gibt anschließend den verminderten Preis zurück.

Häufige Fehler

Argumente in der falschen Reihenfolge übergeben

Bei mehreren Parametern ist die Reihenfolge wichtig.

```
function stelleVor(name, alter) {
  console.log(name + " ist " + alter + " Jahre alt.");
}

stelleVor("Mara", 14);
```

Das ergibt:

```
Mara ist 14 Jahre alt.
```

Wenn du die Werte vertauschst, entsteht ein unpassender Satz:

```
stelleVor(14, "Mara");
```

Ausgabe:

```
14 ist Mara Jahre alt.
```

`return` mit `console.log()` verwechseln

```
function quadriere(zahl) {
  console.log(zahl * zahl);
}
```

Diese Funktion gibt zwar etwas aus, liefert aber kein Ergebnis zurück. Wenn du später mit dem Ergebnis rechnen möchtest, brauchst du `return`:

```
function quadriere(zahl) {
  return zahl * zahl;
```

```
}

let fläche = quadriere(6);

console.log(fläche);
```

Code nach `return` erwarten

```
function teste() {
  return "Fertig";
  console.log("Diese Zeile wird nie ausgeführt.");
}
```

Alles nach `return` innerhalb desselben Funktionsblocks ist nicht erreichbar.

Übung: Temperatur umrechnen

Schreibe eine Funktion `celsiusZuFahrenheit`, die eine Temperatur in Celsius erhält und die passende Temperatur in Fahrenheit zurückgibt.

Zur Umrechnung gilt:

$$
\text{Fahrenheit} = \text{Celsius} \cdot 1{,}8 + 32
$$

Ein möglicher Aufruf soll so aussehen:

```
let temperatur = celsiusZuFahrenheit(20);

console.log(temperatur);
```

Die Ausgabe soll sein:

```
68
```

Lösung anzeigen

```
function celsiusZuFahrenheit(celsius) {
  return celsius * 1.8 + 32;
}

let temperatur = celsiusZuFahrenheit(20);

console.log(temperatur);
```

Übung: Größere Zahl finden

Schreibe eine Funktion `größereZahl`, die zwei Zahlen entgegennimmt und die größere Zahl zurückgibt.

Beispiel:

```
console.log(größereZahl(12, 7));
```

Erwartete Ausgabe:

```
12
```

Eine mögliche Lösung:

```
function größereZahl(zahl1, zahl2) {
  if (zahl1 > zahl2) {
    return zahl1;
  } else {
    return zahl2;
  }
}

console.log(größereZahl(12, 7));
```

Was passiert, wenn beide Zahlen gleich sind? Passe die Funktion so an, dass sie in diesem Fall beispielsweise den Text `"Beide Zahlen sind gleich."` zurückgibt.

Zusammenfassung

- **Parameter** sind Platzhalter für Werte, die eine Funktion beim Aufruf erhält.
- Eine Funktion kann einen oder mehrere Parameter besitzen.
- Mit **Standardwerten** bleibt eine Funktion sinnvoll nutzbar, auch wenn ein Argument fehlt.
- `return` gibt ein Ergebnis aus einer Funktion zurück.
- Rückgabewerte können in Variablen gespeichert, weiterverarbeitet oder in Bedingungen verwendet werden.
- `console.log()` zeigt Werte nur in der Konsole an und ersetzt kein `return`.
- Nach einem `return` wird der restliche Code der Funktion nicht mehr ausgeführt.

Im nächsten Abschnitt nutzt du **Arrays**, um mehrere Werte gemeinsam in einer geordneten Liste zu speichern.

Revision #1
Created 2026-07-24 13:44:21 UTC by art10m
Updated 2026-07-24 13:44:21 UTC by art10m