Funktionen definieren und aufrufen

Warum Funktionen wichtig sind

In Programmen tauchen oft Aufgaben auf, die mehrmals benötigt werden: einen Nutzer begrüßen, einen Preis berechnen oder eine Meldung ausgeben. Statt denselben Code immer wieder zu schreiben, kannst du ihn in einer **Funktion** bündeln.

Eine Funktion ist ein benannter Codeblock, der erst ausgeführt wird, wenn du ihn aufrufst.

```
function begruessen() {
  console.log("Hallo! Schön, dass du da bist.");
}
```

Der Code innerhalb der geschweiften Klammern wird noch nicht automatisch ausgeführt. Zuerst wurde die Funktion nur **definiert**.

Um sie auszuführen, rufst du sie mit ihrem Namen und runden Klammern auf:

```
begruessen();
```

Ausgabe in der Konsole:

```
Hallo! Schön, dass du da bist.
```

> **Merke:** Eine Funktion definieren bedeutet, ihre Anleitung zu schreiben. Eine Funktion aufrufen bedeutet, diese Anleitung auszuführen.

Eine Funktion definieren

Eine Funktionsdefinition besteht aus mehreren Teilen:

```
function zeigeNachricht() {
  console.log("JavaScript macht Webseiten lebendig.");
}
```

Teil	Bedeutung
`function`	Das Schlüsselwort zeigt: Hier beginnt eine Funktion.
`zeigeNachricht`	Der Name der Funktion. Du wählst ihn selbst.
`()`	Hier können später Werte für die Funktion stehen.
`{ ... }`	In diesen geschweiften Klammern steht der auszuführende Code.

Ein Funktionsname sollte möglichst deutlich machen, **was die Funktion tut**. Gute Namen beginnen häufig mit einem Verb:

```
function starteSpiel() {
  console.log("Das Spiel beginnt.");
}

function berechneSumme() {
  console.log(5 + 8);
}

function aendereFarbe() {
  console.log("Farbe wurde geändert.");
}
```

Weniger verständlich wäre dagegen ein Name wie `test()` oder `sache()`. Zwar funktioniert der Code, aber andere Personen — und auch dein zukünftiges Ich — verstehen dann schwerer, wofür die Funktion gedacht ist.

Funktionen aufrufen

Damit JavaScript den Code in einer Funktion ausführt, musst du sie aufrufen:

```
function zeigeUhrzeit() {
  console.log("Es ist Zeit für eine Pause.");
```

```
}

zeigeUhrzeit();
```

Wichtig sind die Klammern beim Aufruf:

```
zeigeUhrzeit();
```

Ohne Klammern schreibst du nur den Namen der Funktion hin:

```
zeigeUhrzeit;
```

Damit wird die Funktion **nicht ausgeführt**. Für Anfänger gilt daher als einfache Regel:

> Wenn eine Funktion jetzt etwas tun soll, schreibe ihren Namen mit `()`.

Du darfst eine Funktion auch mehrfach aufrufen:

```
function klatschen() {
  console.log("□□Klatsch!");
}

klatschen();
klatschen();
klatschen();
```

Die Konsole zeigt dann dreimal:

```
□□Klatsch!
□□Klatsch!
□□Klatsch!
```

Code wiederverwenden

Der größte Vorteil von Funktionen ist die Wiederverwendung. Ohne Funktion müsstest du denselben Code kopieren:

```
console.log("Willkommen im Lernprogramm!");
console.log("Viel Erfolg beim Üben!");

console.log("Willkommen im Lernprogramm!");
console.log("Viel Erfolg beim Üben!");
```

Mit einer Funktion ist der Code ordentlicher:

```
function zeigeWillkommen() {
  console.log("Willkommen im Lernprogramm!");
  console.log("Viel Erfolg beim Üben!");
}

zeigeWillkommen();
zeigeWillkommen();
```

Falls du später den Begrüßungstext ändern möchtest, musst du ihn nur **an einer Stelle** anpassen:

```
function zeigeWillkommen() {
  console.log("Willkommen beim JavaScript-Kurs!");
  console.log("Lerne Schritt für Schritt.");
}
```

Alle Aufrufe von `zeigeWillkommen()` verwenden automatisch die neue Version.

Funktionen können mehrere Anweisungen enthalten

In einer Funktion darf mehr als eine Zeile stehen. JavaScript führt die Anweisungen von oben nach unten aus.

```
function starteProgramm() {
  console.log("Programm wird gestartet.");
  console.log("Daten werden geladen.");
  console.log("Programm ist bereit.");
}
```

```
starteProgramm();
```

Ausgabe:

```
Programm wird gestartet.
Daten werden geladen.
Programm ist bereit.
```

Du kannst in Funktionen auch Variablen verwenden:

```
function zeigePunktestand() {
  const punkte = 120;
  console.log("Dein Punktestand: " + punkte);
}

zeigePunktestand();
```

Funktionen und andere Funktionen

Eine Funktion kann auch eine andere Funktion aufrufen. Das hilft dabei, größere Aufgaben in kleine, gut verständliche Schritte aufzuteilen.

```
function begruessen() {
  console.log("Hallo!");
}

function erklaereAufgabe() {
  console.log("Heute üben wir Funktionen.");
}

function starteLektion() {
  begruessen();
  erklaereAufgabe();
  console.log("Los geht's!");
}

starteLektion();
```

Beim Aufruf von `starteLektion()` passiert Folgendes:

1. `begruessen()` wird ausgeführt.
2. `erklaereAufgabe()` wird ausgeführt.
3. Die letzte Nachricht wird ausgegeben.

So kannst du Programme übersichtlich strukturieren.

Die Reihenfolge von Definition und Aufruf

Bei normalen Funktionsdefinitionen kannst du eine Funktion in JavaScript oft schon vor ihrer Definition aufrufen:

```
sageHallo();

function sageHallo() {
  console.log("Hallo!");
}
```

Der Code funktioniert. Für gut lesbaren Anfänger-Code ist es trotzdem meist übersichtlicher, zuerst die Funktion zu definieren und sie danach aufzurufen:

```
function sageHallo() {
  console.log("Hallo!");
}

sageHallo();
```

Dadurch erkennt man beim Lesen sofort, was die Funktion macht, bevor sie verwendet wird.

Häufige Fehler

Klammern beim Aufruf vergessen

```
function spielen() {
  console.log("Das Spiel läuft.");
}

spielen;
```

Hier erscheint keine Ausgabe, weil `spielen` nicht aufgerufen wurde.

Richtig:

```
spielen();
```

Den Funktionsnamen unterschiedlich schreiben

JavaScript unterscheidet Groß- und Kleinschreibung. `zeigeText` und `ZeigeText` sind zwei unterschiedliche Namen.

```
function zeigeText() {
  console.log("Ein Text");
}

ZeigeText();
```

Das führt zu einem Fehler, weil `ZeigeText` nicht definiert wurde.

Richtig:

```
zeigeText();
```

Geschweifte Klammern vergessen

Der Funktionskörper braucht öffnende und schließende geschweifte Klammern:

```
function testeFunktion() {
  console.log("Test");
}
```

Nicht so:

```
function testeFunktion()
  console.log("Test");
```

Semikolon nach dem Funktionsnamen

Nach der Definition einer klassischen Funktion setzt du kein Semikolon direkt hinter die schließende Klammer des Namens:

```
function zeigeInfo() {
  console.log("Information");
}
```

Ein Semikolon nach `console.log(...)` ist dagegen üblich:

```
console.log("Information");
```

Beispiel: Ein einfaches Mini-Programm

Dieses Beispiel bündelt verschiedene Aufgaben in Funktionen:

```
function zeigeTitel() {
  console.log("=== Zahlenraten ===");
}

function erklaereRegeln() {
  console.log("Rate eine Zahl zwischen 1 und 10.");
}

function verabschiedeDich() {
  console.log("Danke fürs Spielen!");
}

zeigeTitel();
erklaereRegeln();
```

```
verabschiedeDich();
```

Die Funktionen machen das Programm klar gegliedert:

- `zeigeTitel()` kümmert sich um die Überschrift.
- `erklaereRegeln()` zeigt die Spielregel.
- `verabschiedeDich()` beendet das Programm freundlich.

Später könntest du jede Funktion verändern oder erweitern, ohne den restlichen Code stark umzubauen.

Übung: Eigene Funktionen schreiben

Schreibe drei Funktionen:

1. `zeigeName()` gibt deinen Namen in der Konsole aus.
2. `zeigeLieblingsfarbe()` gibt deine Lieblingsfarbe aus.
3. `starteVorstellung()` ruft die beiden anderen Funktionen auf und begrüßt die Person.

Rufe am Ende `starteVorstellung()` auf.

Eine mögliche Lösung:

```
function zeigeName() {
  console.log("Mein Name ist Alex.");
}

function zeigeLieblingsfarbe() {
  console.log("Meine Lieblingsfarbe ist Blau.");
}

function starteVorstellung() {
  console.log("Hallo, ich stelle mich vor:");
  zeigeName();
  zeigeLieblingsfarbe();
}

starteVorstellung();
```

Zusammenfassung

- Eine **Funktion** ist ein benannter Codeblock für eine bestimmte Aufgabe.
- Mit `function` definierst du eine Funktion.
- Mit `funktionsName()` rufst du sie auf.
- Funktionen helfen dir, wiederholten Code zu vermeiden und Programme besser zu ordnen.
- Aussagekräftige Namen wie `zeigeMenue()` oder `starteSpiel()` machen Code leichter verständlich.
- Eine Funktion kann mehrere Anweisungen und auch Aufrufe anderer Funktionen enthalten.

Im nächsten Schritt lernst du, wie Funktionen **Parameter** entgegennehmen und mit `return` Ergebnisse zurückgeben können.

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