Tag 5 - JavaScript: Funktionen

Einleitung: Was sind Funktionen?

Funktionen sind **abgeschlossene Codeblöcke**, die bestimmte Aufgaben erledigen. Sie ermöglichen es, **Code logisch zu gliedern**, wiederverwendbar zu machen und komplexe Programme zu strukturieren.

Vorteile von Funktionen:

- **Wiederverwendbarkeit**: Einmal schreiben, beliebig oft verwenden
- **Struktur**: Gliederung komplexer Programme in kleine, verständliche Teile
- **Wartbarkeit**: Änderungen nur an einer Stelle nötig
- **Lesbarkeit**: Reduzierung von Redundanz erhöht Übersichtlichkeit

---

Verschiedene Möglichkeiten, eine Funktion zu deklarieren

1. Funktionsdeklaration (Function Statement)

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

Diese Art der Deklaration wird beim Parsen des Codes **nach oben gehoben** (hoisting) – man kann sie also auch **vor ihrer Definition aufrufen**.

2. Funktionsausdruck (Function Expression)

```
const begruessung = function() {
  console.log("Hallo!");
};
```

Diese Variante speichert die Funktion in einer Variablen. Sie ist **nicht hoisted**, d.h. sie muss **nach der Definition aufgerufen** werden.

Funktionen aufrufen (Function Call)

Ein Aufruf wird durch den **Funktionsnamen + Klammern** ausgeführt:

```
hallo();
```

Die Klammern signalisieren JavaScript, dass die Funktion **ausgeführt** werden soll.

Man kann eine Funktion beliebig oft aufrufen:

```
hallo();
hallo();
```

Wenn du sie nur deklarierst, passiert nichts. Nur der Aufruf führt den Code aus.

> Tipp: Man kann Funktionen auch in anderen Funktionen aufrufen oder sogar in `console.log()` einbauen.

Lokale Variablen in Funktionen

Was sind lokale Variablen?

Variablen, die **innerhalb** einer Funktion (oder eines Blocks) mit `let`, `const` oder `var` deklariert werden, sind **lokal**. Sie sind **nur in dieser Funktion sichtbar** und **nicht von außen erreichbar**.

Warum ist das wichtig?

- Vermeidet Konflikte mit gleichnamigen Variablen im restlichen Programm
- Macht Funktionen unabhängig und wiederverwendbar

Beispiel:

```
function test() {
  let lokal = "Ich bin nur hier sichtbar";
  console.log(lokal);
}

console.log(lokal); // Fehler: lokal is not defined
```

Gültigkeitsbereich (Scope):

- `let` / `const` → Block-Scope (z. B. Schleifen, if-Blöcke)
- `var` → Function-Scope (gilt in gesamter Funktion)

Praktisches Beispiel:

```
function multipliziere() {
  let zahl = 5;
  let ergebnis = zahl * 2;
  console.log(ergebnis);
}

multipliziere(); // 10
console.log(zahl); // Fehler
```

Rückgabewert mit `return`

Mit dem Schlüsselwort `return` können wir **Werte aus der Funktion zurückgeben** und gleichzeitig den **Ablauf der Funktion sofort beenden**.

Was bewirkt `return`?

- Die Funktion wird **sofort verlassen**, sobald `return` erreicht wird
- Der hinter `return` stehende Ausdruck (z. B. eine Zahl oder ein String) wird an den Aufrufer **zurückgegeben**
- Ist **kein Rückgabewert angegeben**, wird **automatisch** `undefined` zurückgegeben

Warum zeigt die Konsole manchmal `undefined`?

Wenn man in der Browser-Konsole eine Funktion wie `console.log("Hallo")` oder eine eigene Funktion ohne `return` aufruft, sieht man danach `undefined`. Das liegt daran, dass **jede Funktion in JavaScript immer etwas zurückgibt** – und wenn man nichts explizit zurückgibt, ist es eben `undefined`.

Beispiel: ohne Rückgabewert

```
```

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

begruessung(); // Gibt "Hallo!" aus UND zeigt in der Konsole zusätzlich: undefined
```

Beispiel 1: Rückgabe eines Rechenergebnisses

```
function verdoppeln(x) {
  return x * 2;
}

let ergebnis = verdoppeln(6);
console.log(ergebnis); // 12
```

Beispiel 2: Funktion bricht durch `return` frühzeitig ab

```
function debug() {
  console.log("Start");
  return;
  console.log("Dieser Code wird nie erreicht");
}
```

> Wichtig: `return` beendet die Funktion **sofort**.

Parameter und Argumente

Was ist der Unterschied?

- **Parameter**: Platzhalter in der Funktionsdefinition
- **Argumente**: konkrete Werte beim Funktionsaufruf

Beispiel:

```
function begruessung(name) { // ← Parameter
  console.log("Hallo, " + name);
}

begruessung("Lina"); // ← Argument
begruessung("Tarek");
```

Mehrere Parameter:

```
function addiere(a, b) {
  return a + b;
}

console.log(addiere(4, 5)); // 9
```

Parameter verhalten sich wie lokale Variablen innerhalb der Funktion. Wenn gleichnamige globale Variablen existieren, werden sie überschrieben (Shadowing).

Beispiel: Durchschnittstemperatur berechnen

Funktion mit Parameter:

```
function berechneMittelwert(temperaturen) {
  let summe = 0;
  for (let i = 0; i < temperaturen.length; i++) {
    summe += temperaturen[i];
  }
  return summe / temperaturen.length;
}
```

Anwendung:

```
let tag1 = [10, 11, 12];
let tag2 = [20, 22, 24];

console.log(berechneMittelwert(tag1)); // 11
console.log(berechneMittelwert(tag2)); // 22
```

Variablenschatten (Shadowing)

Erklärung:

Wenn Parameter oder lokale Variablen denselben Namen wie eine äußere Variable haben, wird die **lokale Version verwendet**.

Beispiel:

```
let x = 10;
function test(x) {
  let y = 5;
  console.log(x); // Parameter x überschreibt globales x
  console.log(y); // lokale Variable
}
test(1);
console.log(x); // → 10 (globales x bleibt erhalten)
```

Visualisiere den Scope:

```
let a = 100, b = 200, c = 300;
function zeige(a) {
  let b = 10;
  console.log(a); // 1 (Parameter)
  console.log(b); // 10 (lokal)
  console.log(c); // 300 (global)
}

zeige(1);
console.log(a, b, c); // 100, 200, 300
```

Übungen - Funktionen Grundlagen

Aufgabe 1: Einfache Funktion

Erstelle eine Funktion `begruesse()`, die "Guten Morgen!" auf der Konsole ausgibt. Rufe sie drei Mal hintereinander auf.

Aufgabe 2: Rückgabe mit `return`

Schreibe eine Funktion `quadriere(x)`, die eine Zahl quadriert (x * x) und zurückgibt. Gib das Ergebnis auf der Konsole aus.

Aufgabe 3: Parameter und Argumente

Schreibe eine Funktion `begruesse(name)`, die "Hallo, NAME!" ausgibt. Rufe sie mit mindestens 3 verschiedenen Namen auf.

Aufgabe 4: Mittelwert mit Parametern

Erstelle eine Funktion `mittelwert(arr)`, die das arithmetische Mittel eines Zahlen-Arrays zurückgibt. Teste sie mit zwei Arrays.

Aufgabe 5: Funktion mit lokalem Scope

Schreibe eine Funktion, die eine lokale Variable `nachricht` deklariert und diese in der Konsole ausgibt. Zeige, dass `nachricht` außerhalb nicht verfügbar ist.

Aufgabe 6: Globale vs. lokale Variable

Deklariere eine globale Variable `wert = 100`. Erstelle eine Funktion, in der `wert` ebenfalls deklariert ist, aber auf `50` gesetzt wird. Gib in und außerhalb der Funktion `wert` aus und beobachte den Unterschied.

Aufgabe 7: Ohne Rückgabe

Erstelle eine Funktion `zeigeSumme(a, b)`, die zwei Zahlen addiert und das Ergebnis per `console.log()` ausgibt. Rufe sie auf, und beobachte, was die Konsole zusätzlich anzeigt (Stichwort: `undefined`).

Aufgabe 8: Funktionsexpression

Speichere eine Funktion `sagHallo` als Ausdruck in einer Konstante. Die Funktion soll "Hallo!" ausgeben. Rufe die Funktion über die Variable auf.

Aufgabe 9: Parameterreihenfolge

Erstelle eine Funktion `info(name, beruf)`, die einen Satz wie "Lisa ist Webentwicklerin" zurückgibt. Achte beim Aufruf auf die korrekte Reihenfolge der Argumente.

Aufgabe 10: Shadowing testen

Lege globale Variablen `a = 10`, `b = 20` an. Erstelle eine Funktion, in der `a` ein Parameter ist und `b` eine lokale Variable ist. Gib innerhalb der Funktion `a`, `b` und außerhalb `a`, `b` aus.