# JS Bedingungen und Booleans

## Lernziele

- Verwendung von Bedingungen zur Steuerung des Programmflusses
- Verstehen, was Booleans und truthy/falsy Werte sind
- Arbeiten mit Vergleichs- und logischen Operatoren
- Schreiben von ternären Ausdrücken

---

## Boolean-Werte

Ein Boolean-Wert, benannt nach George Boole, hat nur zwei Zustände. Er kann entweder **true** oder **false** sein. Booleans werden oft in bedingten Anweisungen verwendet, die je nach ihrem Wert unterschiedlichen Code ausführen können.

## Truthy- und Falsy-Werte

Manchmal möchte man eine Bedingung basierend auf einem anderen Typ von Wert haben. JavaScript kann jeden Wert durch *Typumwandlung* in einen Boolean verwandeln. Das bedeutet, dass einige Werte sich so verhalten, als wären sie true, und andere, als wären sie false: *Truthy*-Werte werden zu true, *falsy*-Werte werden zu false.

- *truthy* Werte:
  - Zahlen ungleich Null: `1`, `2`, `-3` usw.
  - Nicht leere Strings: `"hello"`
  - `true`
- *falsy* Werte:
  - `0` / `-0`
  - `null`
  - `false`
  - `undefined`
  - Leerer String: `""`

---

## Vergleichsoperatoren

Vergleichsoperatoren erzeugen Boolean-Werte, indem sie zwei Ausdrücke vergleichen:

| Operator | Wirkung |
|---|---|
| A `===` B | strikt gleich: ist `true`, wenn beide Werte gleich sind (einschließlich ihres Typs). |
| A `!==` B | strikt ungleich: ist `true`, wenn beide Werte nicht gleich sind (einschließlich ihres Typs). |
| A `>` B | strikt größer als: ist `true`, wenn A größer als B ist. |

| Operator | Wirkung |
|---|---|
| A < B | strikt kleiner als: ist `true`, wenn A kleiner als B ist. |
| A >= B | größer als oder gleich: ist `true`, wenn A größer als oder gleich B ist. |
| A <= B | kleiner als oder gleich: ist `true`, wenn A kleiner als oder gleich B ist. |

💡 Du wirst vielleicht bemerken, dass JavaScript drei Gleichheitszeichen (`===`) verwendet, um auf Gleichheit zu prüfen. Das kann anfangs sehr ungewöhnlich erscheinen.

- `=` (`const x = 0`) ist der Zuweisungsoperator und hat nichts mit Vergleich zu tun.
- `==` und `!=` sind nicht-strikte Gleichheitsoperatoren. Du solltest **sie 99% der Zeit vermeiden**. Nicht-strikte Gleichheit versucht, beide Werte mithilfe von Typumwandlung in denselben Typ zu konvertieren: `"3" == 3` ergibt `true`, was selten das ist, was du willst.
- `===` und `!==` sind strikte Gleichheitsoperatoren. **Das ist fast immer das, was du brauchst**. Strikte Gleichheit prüft, ob Typ *und* Wert gleich sind: `"3" === 3` ergibt `false`.

---

## Logische Operatoren

Logische Operatoren kombinieren bis zu zwei Booleans zu einem neuen Boolean.

| Operator | Wirkung |
|---|---|
| !A | `not`: kehrt einen `true` Wert in `false` um und umgekehrt. |
| A \|\| B | `or`: ist `true`, wenn entweder A `oder` B `true` ist. |
| A && B | `and`: ist `true`, wenn sowohl A `und` B `true` sind. |

💡 Du kannst logische Operatoren mit Klammern kombinieren, um festzulegen, welcher Operator zuerst ausgewertet werden soll, z.B.:

- `(A || B) && (C || D)`
- `!(A || B)`

💡 Sei vorsichtig beim Verwenden von `&&` oder `||` mit nicht-boolean Werten. Sie geben tatsächlich einen der ursprünglichen Werte zurück. Das kann nützlich sein, kann aber auch schnell zu Verwirrung führen. Dieses Verhalten wird als Kurzschlussbewertung bezeichnet und ist ein fortgeschrittenes Thema.

- `"some string" || "some other string"` ergibt `"some string"`
- `0 || 100` ergibt `100`
- `null && "yet another string"` ergibt `null`

---

## Steuerfluss: `if / else`

Mit einer if-Anweisung können wir steuern, ob ein Teil unseres Codes ausgeführt wird oder nicht, basierend auf einer Bedingung.

```
const isSunShining = true;

if (isSunShining) {
  // Code, der nur ausgeführt wird, wenn die Bedingung "isSunShining" true
ist
}
```

Der else-Block wird nur ausgeführt, wenn die Bedingung false ist.

```
const isSunShining = false;

if (isSunShining) {
  // Code, der nur ausgeführt wird, wenn die Bedingung "isSunShining" true
ist
} else {
  // Code, der nur ausgeführt wird, wenn die Bedingung "isSunShining"
false ist
}
```

Der Bedingungsausdruck zwischen den `()` Klammern kann ebenfalls aus logischen oder Vergleichsoperatoren bestehen. Du kannst zwischen mehreren Fällen unterscheiden, indem du `else if`-Anweisungen verkettest:

```
if (hour < 12) {
  console.log("Guten Morgen.");
} else if (hour < 18) {
  console.log("Guten Nachmittag.");
} else if (hour === 24) {
  console.log("Gute Nacht.");
} else {
  console.log("Guten Abend.");
}
```

Wenn die Bedingung kein Boolean ist, wird sie durch Typumwandlung in einen Boolean konvertiert. Dies kann verwendet werden, um zu überprüfen, ob ein Wert nicht 0 oder ein leerer String ist:

```
const name = "Alex";
if (name) {
  console.log("Hi " + name + "!"); // wird nur ausgeführt, wenn name nicht
ein leerer String ist
}
```

## Switch

Manchmal möchten wir eine Variable oder einen Ausdruck überprüfen, um zu sehen, ob ihr Wert einer von wenigen sehr spezifischen möglichen Werten ist. In diesem Fall können wir die **switch**-Anweisung verwenden.

Die **switch**-Anweisung funktioniert, indem sie die Variable oder den Ausdruck nimmt und dann nacheinander durch eine Liste von möglichen **Fällen** für ihren Wert geht. In jedem `case`, den sie erreicht, vergleicht sie den überprüften Wert mit dem Wert des `case`. Wenn sie übereinstimmen, wird der Code in diesem `case` ausgeführt. Wenn sie nicht übereinstimmen, wird JavaScript zum nächsten `case` in der Reihenfolge weitergehen.

Hier ein Beispiel:

```
console.log("Was ist deine Lieblingsjahreszeit?");
const userAnswer = "Frühling";

switch (userAnswer) {
  case "Sommer":
    console.log("Hitze, Sonne und Wellen für dich 😎");
    break;
  case "Herbst":
    console.log("Knisprige, bunte Blätter und kühle Brisen 🍁");
    break;
  case "Winter":
    console.log("Eis, Schnee, warme Kleidung und heiße Getränke ☕");
    break;
  case "Frühling":
    console.log("Wachstum, Grün und neue Anfänge! 🌿");
    break;
  default:
    console.log(
      "Entschuldigung, ich glaube nicht, dass das eine Jahreszeit ist!"
    );
}
```

Wichtige Hinweise zur `switch`-Anweisung:

1. Wie bei einer if-Anweisung musst du den überprüften Wert/Ausdruck in Klammern setzen
2. Wie bei einer if-Anweisung, den gesamten Block der switch-Anweisung in geschweifte Klammern setzen
3. Jeder case bietet einen einzelnen Wert (nicht einen Ausdruck!) zur Überprüfung an
4. Nach dem Wert des case muss ein Doppelpunkt (😃 stehen
5. Um zu verhindern, dass JavaScript von einem case zum nächsten fließt, stelle sicher, dass jeder case mit einem break endet!
6. Du kannst einen default-Fall hinzufügen, wenn du möchtest; das ist generell eine gute Idee. Er muss nicht mit break enden

## Ternärer Operator: `? :``

Mit if / else-Anweisungen kann man ganze Codeblöcke steuern. Der ternäre Operator kann verwendet werden, wenn du zwischen zwei Ausdrücken entscheiden möchtest, z.B. welcher Wert in einer Variable gespeichert werden soll:

```
const greetingText = time < 12 ? "Guten Morgen." : "Guten Nachmittag.";
```

Der ternäre Operator hat die folgende Struktur:

```
condition ? expressionIfTrue : expressionIfFalse;
```

Wenn die Bedingung true ist, wird der erste Ausdruck ausgewertet, andernfalls der zweite Ausdruck. Der ternäre Operator kann verwendet werden, um zu entscheiden, welche Funktion aufgerufen werden soll:

```
isUserLoggedIn ? logoutUser() : loginUser();
```

Er kann auch entscheiden, welcher Wert als Argument an eine Funktion übergeben werden soll:

```
moveElement(xPos > 300 ? 300 : xPos); // das Element kann nicht weiter als
300 bewegt werden.
```

> ❗ Der Operator kann nur zwischen zwei *Ausdrücken* wie Werten, mathematischen/logischen Operationen oder Funktionsaufrufen unterscheiden, nicht zwischen *Anweisungen* wie Variablendeklarationen, if / else-Anweisungen oder mehrzeiligen Codeblöcken.

---

## Fortgeschritten: Die Eigenart der Boolean-Koerzierung und die Nutzung nicht-strikter Gleichheit

▶ 😲 Dies ist ein fortgeschrittenes Thema und nicht wichtig für die Herausforderungen. Klicke hier, wenn du neugierig bist.

Angenommen, du möchtest überprüfen, ob eine Variable einen nützlichen Wert für uns hat. `if(variable)` prüft tatsächlich nicht, ob `variable` definiert ist, sondern ob es truthy ist. Sieh dir diese Beispiele an:

- if(undefined) → falsy, wird nicht ausgeführt
- if(null) → falsy, wird nicht ausgeführt
- if("") → falsy, wird nicht ausgeführt, könnte aber trotzdem eine nützliche Variable sein (z.B. wenn der Benutzer ein Eingabefeld leert)
- if(0) → falsy, wird nicht ausgeführt, könnte aber trotzdem eine nützliche Variable sein (z.B. wenn der Benutzer den Lautstärkepegel auf 0 setzen möchte)
- if(" ") → truthy, wird ausgeführt
- if(-1) → truthy, wird ausgeführt

Es ist nützlich, eine Variable als nicht vorhanden zu definieren, wenn sie `undefined` oder `null` ist. Wir können dies so überprüfen:

```
if (variable != null) {
  console.log('Das wird protokolliert, selbst wenn die Variable 0 oder ""
ist');
}
```

Dies ist einer der seltenen gültigen Anwendungsfälle für nicht-strikte Vergleiche (`!=` statt `!==`).

JavaScript versucht, die verglichenen Werte in denselben Typ zu zwingen. Und genauso wie `"3" == 3` `true` ist, ist auch `undefined == null` `true`. Das funktioniert auch mit `!=` statt `==`.

> ⚠ Denke daran, dass dies eine Ausnahme für die Verwendung nicht-strikter Gleichheit ist. **Strikte Gleichheit sollte ansonsten immer bevorzugt werden.**

---

## Ressourcen

### Operatoren

MDN Comparison Operators

MDN Logical Operators

### if / else-Anweisungen

MDN about if else

### Ternärer Operator

MDN Ternary Operator