Tag 3 - JavaScript: Operatoren

Einleitung

Operatoren sind Symbole (oder Schlüsselwörter), die bestimmte Operationen mit Werten oder Variablen – den sogenannten **Operanden** – durchführen. In JavaScript gibt es:

- **Unäre Operatoren** (1 Operand, z. B. `typeof`)
- **Binäre Operatoren** (2 Operanden, z. B. `+`, `-`, `==`)
- **Ternäre Operatoren** (3 Operanden, z. B. der ?-Operator)

Je nach ihrer Position spricht man auch von **Prefix-, Infix-** oder **Postfix-Operatoren**:

- Infix: `a + b`
- Prefix: `++a`
- Postfix: `a++`

In der Praxis kategorisiert man sie meist nach Funktion:

- Zuweisungsoperatoren
- Arithmetische Operatoren
- Vergleichsoperatoren
- Logische Operatoren
- Weitere Spezialoperatoren (`typeof`, `instanceof`, `delete`)

Wichtig: Ein und dasselbe Symbol kann je nach **Datentyp der Operanden** etwas völlig anderes bedeuten! Das `+`-Zeichen ist dafür das beste Beispiel:

```
2 + 3       // → 5 (Zahlen: Addition)
"A" + "B"   // → "AB" (Strings: Verkettung)
```

---

1. Zuweisungsoperatoren

Der einfachste Operator ist der **Zuweisungsoperator** `=`.

```
const name = "Alice";
```

Er weist dem Namen auf der linken Seite den Wert auf der rechten Seite zu.

Reihenfolge bei mehrfacher Zuweisung (rechtsassoziativ)

```
let year = 2050;
let newYear = year = 2051;
console.log(newYear); // → 2051
```

Hier wird zuerst `year = 2051` ausgeführt, danach `newYear = year`.

2. Arithmetische Operatoren (binär)

Operator	Bedeutung	Beispiel	Ergebnis
`+`	Addition / Verkettung	`2 + 3`, `"A" + "B"`	`5`, "AB"
`-`	Subtraktion	`7 - 2`	`5`
`*`	Multiplikation	`3 * 4`	`12`
`/`	Division	`10 / 2`	`5`
`%`	Modulo	`10 % 3`	`1`
`**`	Potenzieren	`2 ** 3`	`8`

Beispiel:

```
const x = 5;
const y = 2;
console.log("addition:", x + y);          // 7
console.log("subtraction:", x - y);       // 3
console.log("multiplication:", x * y);    // 10
console.log("division:", x / y);          // 2.5
console.log("remainder:", x % y);         // 1
console.log("exponent:", x ** y);         // 25
```

Wichtig:

- Nur der `+`-Operator kann auch Strings verarbeiten.
- Alle anderen Operatoren konvertieren implizit in `number`.
- **Klammern** können die Reihenfolge der Ausführung steuern:

```
console.log(2 + 2 * 2);       // 6
console.log((2 + 2) * 2);     // 8
```

3. Arithmetische Operatoren (unär)

Unäres Plus und Minus:

Diese Operatoren konvertieren den Wert in eine Zahl – `-` negiert zusätzlich.

```
let str = "123";
let n1 = +str;     // 123
let n2 = -str;     // -123
let n3 = -n2;      // 123
let n4 = +"abcd";  // NaN
```

Typprüfung dabei:

```
console.log(typeof str); // "string"
console.log(typeof n1);  // "number"
```

4. Inkrement und Dekrement

Diese unären Operatoren verändern den Wert um ±1. Es gibt zwei Schreibweisen:

Postfix:

```
let a = 1;
console.log(a++); // 1 (zeigt zuerst alten Wert)
console.log(a);   // 2
```

Präfix:

```
let b = 1;
console.log(++b); // 2 (zeigt neuen Wert direkt)
```

Besonderheit:

Das Verhalten wirkt ähnlich, ist aber wichtig bei komplexeren Ausdrücken oder Schleifen.

---

5. Kombinierte Zuweisungsoperatoren

Operator	Entspricht
`+=`	`a = a + b`
`-=`	`a = a - b`
`*=`	`a = a * b`
`/=`	`a = a / b`
`%=`	`a = a % b`
`**=`	`a = a ** b`

Beispiel:

```
let x = 10;
x += 2; // 12
x -= 4; // 8
x *= 3; // 24
x /= 6; // 4
x **= 3; // 64
x %= 10; // 4
```

6. Logische Operatoren

Logische Operatoren arbeiten mit **Booleschen Werten** (`true` oder `false`). In JavaScript gibt es drei grundlegende logische Operatoren:

Operator	Bedeutung	Beispiel	Ergebnis
`&&`	Logisches UND	`true && false`	`false`
`||`	Logisches ODER	`true || false`	`true`
`!`	Logisches NICHT	`!true`	`false`

Konkrete Beispiele:

```
console.log(true && true);   // true
console.log(true && false);  // false
console.log(false || true);  // true
console.log(!false);         // true
```

7. Operator-Priorität bei logischen Ausdrücken

Wie bei Rechenoperationen entscheidet auch hier die Reihenfolge über das Ergebnis. Die **Reihenfolge (Präzedenz)** lautet:

1. `!` (NOT)
2. `&&` (AND)
3. `||` (OR)

Klammern können wie immer verwendet werden, um die Auswertung explizit zu steuern.

```
const a = false;
const b = true;
const c = false;
const d = true;

console.log(a && b && c || d);       // true
console.log(a && b && (c || d));     // false
```

8. Logische Operatoren mit Nicht-Boolean-Werten

JavaScript ist flexibel – logische Operatoren können auch mit anderen Datentypen verwendet werden. Dabei wird intern eine **implizite Typumwandlung** vorgenommen (coercion).

Logisches NICHT `!`

```
let nr = 0;
let year = 1970;
let name = "Alice";
let empty = "";

console.log(!nr);     // true  → 0 ist falsy
console.log(!year);   // false → 1970 ist truthy
console.log(!name);   // false → nicht leer
console.log(!empty);  // true  → leerer String ist falsy
```

Zwei Ausrufezeichen (`!!`) konvertieren einen Wert explizit zu `true` oder `false`:

```
console.log(!!name); // true
```

`&&` und `||` geben keinen Boolean zurück!

`&&` (UND)

- Gibt den **ersten falsy Wert** zurück
- Oder, falls keiner falsy ist, den letzten Wert

```
console.log(true && 1991);         // 1991
console.log(false && 1991);        // false
console.log("Alice" && "Bob");    // "Bob"
console.log(0 && "Bob");          // 0
```

`||` (ODER)

- Gibt den **ersten truthy Wert** zurück
- Oder den letzten, wenn alle falsy sind

```
console.log(true || 1991);         // true
console.log(false || 1991);        // 1991
console.log("" || "Bob");         // "Bob"
```

---

9. Short-Circuit Evaluation

Logische Operatoren **überspringen** die Auswertung des zweiten Operanden, wenn dieser nicht mehr benötigt wird.

Beispiel:

```
let x = 0;
let y = 0;
console.log(x++ && y++); // 0
console.log(x); // 1
console.log(y); // 0 → wurde nicht ausgeführt!
```

Diese Eigenschaft kann in der Praxis sinnvoll, aber auch fehleranfällig sein. Besonders bei Nebenwirkungen (z. B. `y++`) ist Vorsicht geboten.

---

10. Kombinierte logische Zuweisungsoperatoren

Wie bei arithmetischen Operatoren gibt es auch bei logischen Operatoren **kombinierte Varianten**:

Operator	Entspricht
`&&=`	`a = a && b`
`||=`	`b = b || true`

Beispiel:

```
let a = true;
a &&= false;
console.log(a); // false
```

```
let b = false;
b ||= true;
console.log(b); // true
```

---

11. String-Operatoren

In JavaScript gibt es nur **einen echten String-Operator**: den **+-Operator zur Verkettung** (Konkatenation).

Beispiel:

```
let begruessung = "Hi";
console.log(begruessung + " " + "Alice"); // → "Hi Alice"
```

Wenn einer der Operanden ein **String** ist, wird der andere automatisch zu einem String konvertiert.

```
let satz = "Happy New Year ";
let neueNachricht = satz + 10191;
console.log(neueNachricht); // → "Happy New Year 10191"
console.log(typeof neueNachricht); // → "string"
```

Kombinierte Zuweisung mit Strings

```
let text = "Happy New ";
text += "Year ";
text += 10191;
console.log(text); // → "Happy New Year 10191"
```

---

12. Vergleichsoperatoren

Vergleichsoperatoren vergleichen zwei Werte und liefern immer ein **Boolean-Ergebnis** (`true` oder `false`).

Operator	Bedeutung	Typprüfung	Beispiel	Ergebnis
`==`	Gleich (Wert)	❌ Nein	`10 == "10"`	`true`
`===`	Strikt gleich	✅ Ja	`10 === "10"`	`false`
`!=`	Ungleich (Wert)	❌ Nein	`0 != false`	`false`
`!==`	Strikt ungleich	✅ Ja	`0 !== false`	`true`
`>`	Größer	nach Wert	`101 > 100`	`true`
`<`	Kleiner	nach Wert	`"10" < 20`	`true`
`>=`	Größer gleich	nach Wert	`10 >= 10n`	`true`
`<=`	Kleiner gleich	nach Wert	`"10" <= 20`	`true`

Wichtig:

- `===` und `!==` vergleichen auch den **Datentyp**.
- `==` konvertiert Werte ggf. automatisch (z. B. `"10"` → `10`)
- Vergleiche mit `NaN` sind immer **false**, auch `NaN == NaN`

Beispiele:

```
console.log(10 === 10);       // true
console.log(10 === "10");    // false
console.log(0 == false);      // true
console.log(undefined == false); // false
console.log(NaN == NaN);      // false
```

String-Vergleich:

Vergleiche erfolgen Zeichen für Zeichen (Unicode-Wert)

```
console.log("b" > "a");     // true
console.log("A" < "a");     // true (Großbuchstaben haben niedrigeren Wert)
```

---

13. Der ternäre Operator

Der ternäre Operator ist der einzige JavaScript-Operator mit **drei Operanden**. Er ist eine Kurzform für `if...else`.

Syntax:

```
bedingung ? wertWennTrue : wertWennFalse
```

Beispiel:

```
let name = true ? "Alice" : "Bob";  // → "Alice"
let wert = 1 > 2 ? "Ja" : "Nein";   // → "Nein"
```

Er eignet sich für einfache Bedingungen, vor allem beim Zuweisen von Werten.

---

14. Operatorpräzedenz und Assoziativität

Die Reihenfolge, in der Operatoren ausgewertet werden, wird durch **Präzedenz (Vorrang)** und **Assoziativität** geregelt.

Beispiel:

```
let a = 10;
let b = a + 2 * 3;           // → 16
let c = a + 2 < 20 - 15;     // → false
```

Klammern helfen!

```
let d = (20 + 20) * 2;       // 80
let e = d > (3 ** 2);        // true
```

Wichtig:

- **Assoziativität** ist die Richtung:
  - `+`, `*`, `<` etc. → **linksassoziativ** (`a + b + c` = `(a + b) + c`)
  - `=` und `? :` → **rechtsassoziativ** (`a = b = 5` = `a = (b = 5)`)

Die Kombination aus Präzedenz und Assoziativität beeinflusst das Ergebnis, besonders bei komplexen Ausdrücken.

Beispiel:

```
let a, b;
b = (a = (20 + 20) * 2) > (3 ** 2);
console.log(a); // 80
console.log(b); // true
```

Regel: Wenn du unsicher bist – **Klammern setzen**! Sie erhöhen nicht nur die Kontrolle über die Auswertung, sondern auch die Lesbarkeit deines Codes.

15. Übungen zu Operatoren

Aufgabe 1: Arithmetische Operatoren

Erstelle zwei Variablen `x = 10` und `y = 3`.

- Berechne: Summe, Differenz, Produkt, Quotient, Modulo, Potenz
- Gib alle Ergebnisse mit `console.log()` aus.

Aufgabe 2: Unäre Operatoren testen

- Erstelle eine Variable mit dem String "123"
- Verwandle sie mit `+` in eine Zahl und negiere sie mit `-`
- Überprüfe den Typ mit `typeof`

Aufgabe 3: Inkrement vs. Dekrement

- Erstelle eine Variable `z = 5`
- Gib den Wert vor und nach Anwendung von `z++` und `++z` aus
- Nutze zusätzlich `z--` und `--z` zur Verdeutlichung

Aufgabe 4: Kombinierte Zuweisungen

- Erstelle `let a = 10`
- Nutze der Reihe nach `+=`, `-=`, `*=`, `/=`, `%=` und `**=` mit verschiedenen Werten
- Kommentiere die Ergebnisse

Aufgabe 5: Logische Operatoren mit Wahrheitswerten

- Teste Kombinationen mit `&&`, `||` und `!` anhand von `true` und `false`
- Dokumentiere alle Kombinationen (z. B. `true && false`)

Aufgabe 6: Logik mit Nicht-Boolean-Werten

- Was ergibt `0 && "Hallo"`?
- Was ergibt `"" || "Fallback"`?
- Was ergibt `!!"Text"`?

Aufgabe 7: Vergleichsoperatoren

- Erstelle Vergleiche mit `==`, `===`, `!=`, `!==`
- Verwende unterschiedliche Typen (String, Zahl, Boolean)
- Nutze z. B. `"5" == 5`, `null == undefined`, `NaN == NaN`

Aufgabe 8: Stringverkettung & Typen

- Erstelle `let satz = "Das Ergebnis ist: "`
- Verkette mit einer Zahl, einem Boolean, und einem weiteren String
- Überprüfe den Typ des Gesamtergebnisses

Aufgabe 9: Ternärer Operator in Aktion

- Frage den Benutzer per `prompt()` nach einem Alter
- Nutze `? :`, um abhängig vom Alter "Volljährig" oder "Minderjährig" auszugeben

Aufgabe 10: Operatorpräzedenz erkennen

```
let a = 5 + 2 * 4;
let b = (5 + 2) * 4;
let c = 5 + 2 < 10 - 3;
console.log(a, b, c);
```

- Erkläre mit Kommentaren, warum die Ergebnisse so sind wie sie sind