JavaScript 2

Variablen, Operatoren, Ein- und Ausgabe

Werte speichern

Im Programmablauf müssen immer wieder verschiedene Werte gespeichert werden, damit auf sie zugegriffen werden kann

Soll ein Programm etwa zwei Zahlen addieren, geht das z.B. so:

- Lies die erste Zahl und merke sie dir
- Lies die zweite Zahl und merke sie dir
- Addiere die beiden gemerkten Zahlen
- Merke dir das Ergebnis
- Zeige mir das Ergebnis

Damit das Programm auf die gemerkten Werte zugreifen kann, bekommen sie einen eindeutigen Namen („Bezeichner“), etwa so:

- Lies die erste Zahl und merke sie dir unter dem Namen „ersteZahl“

Variablen

Alle Werte, mit denen ein Programm arbeitet, werden gewissermaßen in Kästchen gepackt, die eine eindeutige Beschriftung haben (= einen Namen)

Diese Kästchen sind die „Variablen“ (es gibt auch „Konstanten“)

Auf den Inhalt der Variablen greift man über ihren Namen zu

Der aktuelle Inhalt des Kästchens ist der *Wert* einer Variable

Der Inhalt des Kästchens kann sich im Programmverlauf beliebig ändern

Damit ändert sich auch der Wert, für den eine Variable steht

Variablen deklarieren

Bevor eine Variable benutzt werden kann, muss sie „deklariert“ werden (das Kästchen muss bereitgestellt und beschriftet werden)

Variablen können auf verschiedene Weise deklariert werden, wir nehmen nur `let`:

```
let antwortAufAlles;
```

Damit wird die Variable antwortAufAlles bereitgestellt, sie ist aber noch leer (hat also keinen Wert / Inhalt). Eine Typangabe ist nicht notwendig

Die Wertzuweisung erfolgt über das Gleichheitszeichen, z.B. so:

```
let antwortAufAlles = 42;
```

Hier wird also die Variable `antwortAufAlles` deklariert und ihr gleichzeitig der Wert `42` zugwiesen

Beispiele für Wertzuweisungen

Häufigste Form ist die einfache Wertzuweisung:

```
let mehrwertsteuer = 19;
```

Es können auch Berechnungen durchgeführt werden:

```
let summe = 17 + 4;
```

```
let a = 5; let b = 2;
```

```
let c = a + b;
```

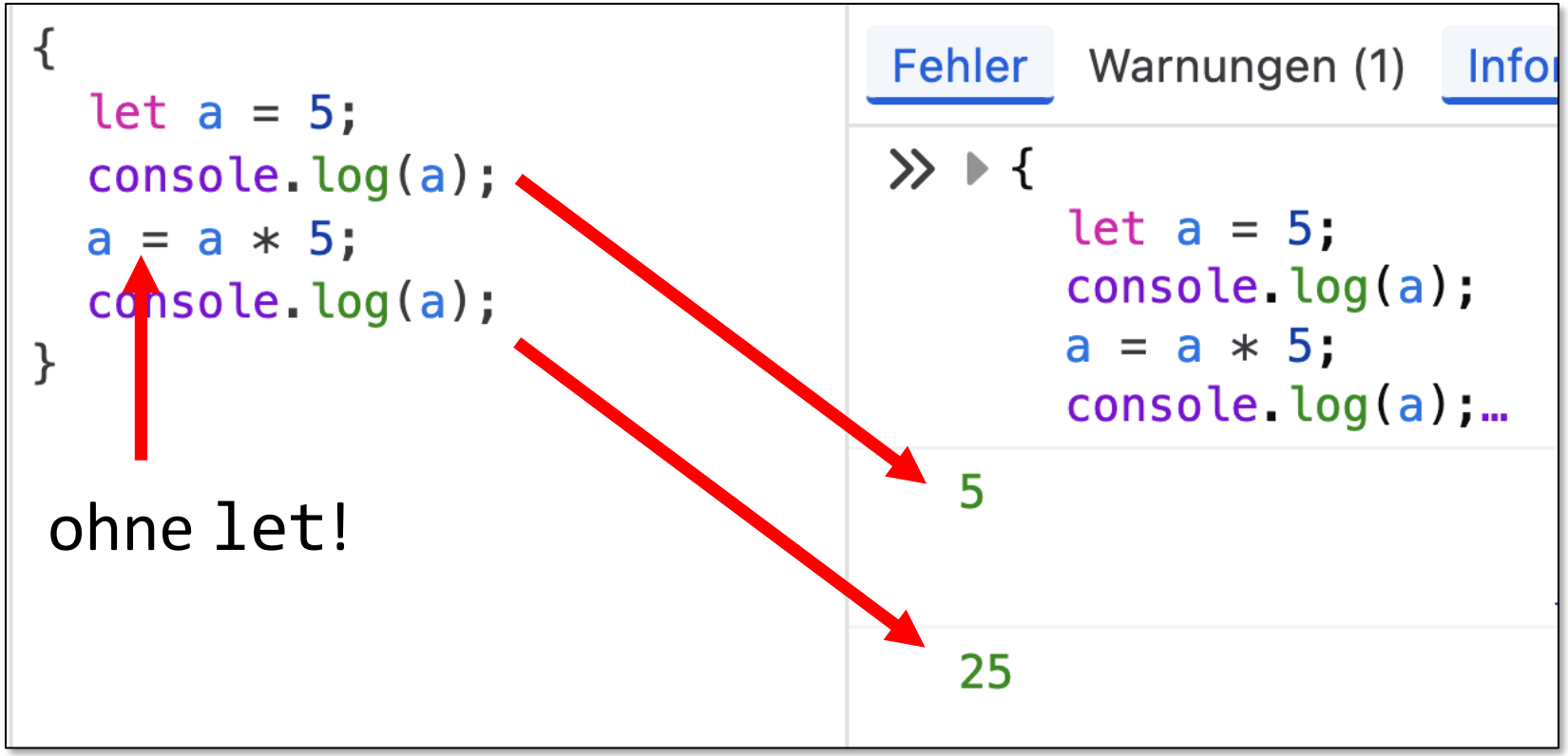
Sehr oft benötigt man so etwas:

```
a = a + 3;
```

D.h.: nimm den aktuellen Wert von a, addiere 3 und weise das Ergebnis der Variablen a als neuen Wert zu

Variablen aufrufen

Sobald eine Variable deklariert wurde, kann sie über ihren Namen aufgerufen werden. An die Stelle des Namens wird dann der aktuelle Wert der Variable eingefügt. Das Schlüsselwort `let` wird nur bei der Deklaration benutzt, danach nur noch der Variablenname.



Variablen benennen

Der Name einer Variablen kann frei gewählt werden

Es empfiehlt sich, sprechende Namen zu benutzen. Statt eines nichtssagenden `x` also eher `eingabe`

Es gibt einige Benennungsregeln (die sind nicht typisch für JavaScript, sondern gelten für praktisch alle Programmiersprachen):

- Nicht erlaubt sind Leerzeichen, Satzzeichen und führende Ziffern
- Zeichen und Wörter, die in JS eine besondere Bedeutung haben („Token“), sind ebenfalls nicht möglich

Best practice: Beschränkung auf `a` bis `Z`, `0` bis `9` und `_`

Variablen: Falsch und richtig

`let irgendwas;`	**Korrekt:** legt eine nicht definierte Variable `irgendwas` an
`let var1 = "Test";`	**Korrekt:** legt `var1` mit dem Inhalt `Test` an
`let 1variable = "Test";`	**Falsch:** führende Ziffern sind nicht erlaubt
`let let = "Test";`	**Falsch:** `let` ist ein reserviertes Schlüsselwort
`let varEins = "Test";`	**Korrekt:** legt `varEins` mit dem Inhalt `Test` an
`let var Eins = "Test";`	**Falsch:** Leerzeichen sind nicht erlaubt
`let var!Eins = "Test";`	**Falsch:** Satzzeichen sind nicht erlaubt
`let varEins = "Test";` `let varEins = "Text";`	**Falsch:** Variablen, die mit `let` deklariert wurden, können nicht erneut deklariert werden

Exkurs: Variablen-Scope

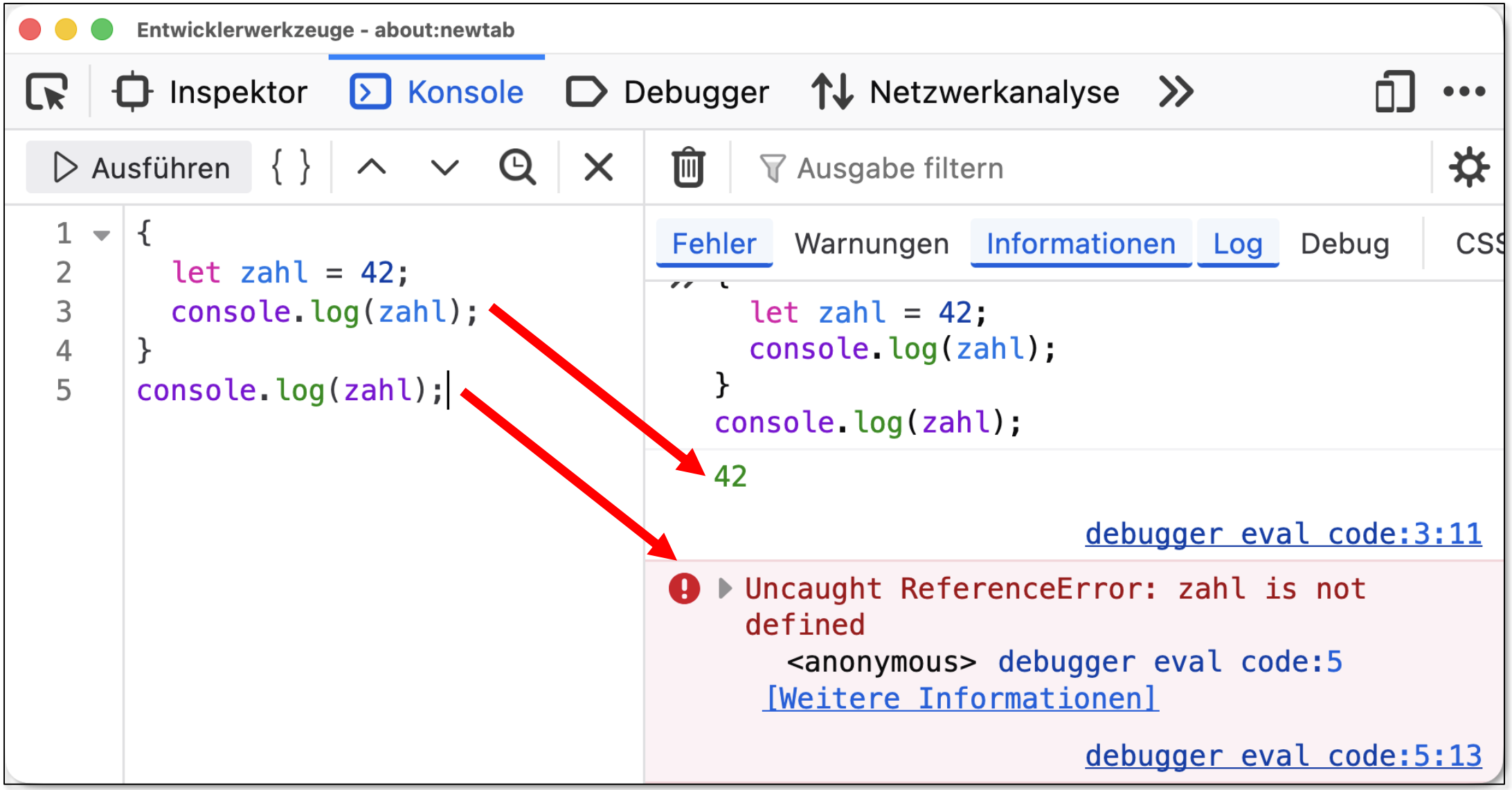
Variablen sind nicht „überall“, sondern nur in bestimmten Programmabschnitten „sichtbar“

Der „Scope“ einer Variable gibt an, in welchen Programmbereichen auf sie zugegriffen werden kann

Ein weites Feld – wir machen uns das ganz einfach. Faustregel:

- Eine in einem Block mit `let` deklarierte Variable ist auch nur in diesem Block sichtbar (ein Block wird von Mengenklammern umschlossen `{ }`)
- Wird versucht, auf eine nicht-sichtbare Variable zuzugreifen, erfolgt eine Fehlermeldung

Beispiel: Variablen-Scope



Konstanten

Konstanten sind im Prinzip Variablen (also benannte Container, denen ein Wert zugewiesen wird), aber …

… im Unterschied zu Variablen lässt sich ein einmal zugewiesener Wert *nicht* mehr ändern (*variabel* vs. *konstant* …)

- Konstanten werden mit dem Schlüsselwort `const` deklariert
- Konstanten muss ein Wert zugewiesen werden
- `const` ist wie `let` blockorientiert (ein „Block“ wird von Mengenklammern `{ }` umschlossen)

Konstanten: Beispiele

```
const player1 = "Peter";
```

Korrekt: Reserviert Speicherplatz unter dem Namen `player1` und weist ihm den festen Wert `Peter` zu. Lässt sich nach der Deklaration nicht mehr ändern

```
const player1;
```

Fehler: Der Konstanten muss ein Wert zugewiesen werden

```
const player1 = "Peter";
player1 = "Petra";
```

Fehler: Einmal deklariert kann der Konstanten kein neuer Wert zugewiesen werden

```
const player1 = "Peter";
let p1 = player1;
p1 = "Petra";
```

Korrekt: Zuerst wird die Konstante `player1` angelegt. Anschließend wird der Inhalt der Konstanten an die Variable `p1` übertragen. Der Inhalt von `p1` kann beliebig geändert werden.

Datentypen

Je nach Inhalt hat eine Variable einen bestimmten Datentyp. Der Datentyp bestimmt, welche Werte in einer Variablen gespeichert und wie eine Variable eingesetzt werden kann. Zahlen können z.B. multipliziert werden – Texte nicht

Programmiersprachen gehen bei der Typisierung unterschiedliche Wege, generell lassen sich zwei Verfahren unterscheiden:

- **Starke / strenge Typisierung:** Bevor eine Variable benutzt werden kann, muss ihr Datentyp explizit festgelegt werden. Einmal festgelegt, lässt sich der Datentyp nicht mehr ändern.
- **Schwache / dynamische Typisierung:** Der Datentyp einer Variablen wird automatisch festgelegt, sobald ihr ein Wert zugewiesen wird. Eine Variable kann ihren Datentyp jederzeit ändern, entweder explizit durch „Typecasting“ oder automatisch durch den Programmablauf.

Starke vs. schwache Typisierung

Die starke Typisierung ist aufwendig, komplex und starr, bietet dafür aber Vorteile:

- Der Programmcode kann optimal in Maschinencode übersetzt werden. Der endgültige Programmcode wird kompakter und kann in der Regel schneller abgearbeitet werden
- Fehler fallen sofort auf, etwa wenn einer numerischen Variablen Text zugewiesen wird oder eine ganze Zahl in einem Kontext benutzt wird, der Fließkommazahlen benötigt

Schwache Typisierung ist dagegen zwar fehleranfälliger und produziert unter Umständen nicht so ganz optimalen Programmcode, ist aber einfacher, schneller und flexibler

Datentypen in JavaScript

JavaScript ist eine schwach bzw. dynamisch typisierende Sprache

Es ist nicht erforderlich, einen Datentyp anzugeben – der wird automatisch erzeugt

„Dynamisch“ ist die Typisierung in JavaScript, weil der Datentyp automatisch angepasst wird und sich im Programmablauf auch ändern kann

Aus einer Zahl kann also eine Zeichenkette (String) werden, aus einem logischen Wahrheitswert eine Zahl

Das erleichtert vieles, kann aber auch zu fiesen Fehlern führen …

Die wichtigsten Datentypen in JS

`Number`	Für numerische Werte, also Zahlen. Es wird nicht zwischen ganzen Zahlen (*Integer*) und Fließkommazahlen (*Float*) unterschieden. Kommazahlen werden wie gewohnt mit einem Punkt notiert: 3.14
`String`	Zeichenketten, also beliebiger Text. Strings stehen in `"`doppelten`"` oder `'`einfachen`'` Anführungszeichen.
`Boolean`	Logischer Wahrheitswert (benannt nach George Boole). Eine Variable vom Typ Boolean kann entweder `true` oder `false` enthalten. Diese Variablen werden für Entscheidungen im Programmablauf benötigt.
`Object`	Für alles, was irgendwie mit Objekten zu tun hat … ;-) (hiermit lassen sich eigene Datentypen definieren)
`undefined` `null`	Sonderformen. Eine Variable ist `undefined`, wenn ihr kein Wert zugewiesen wurde, sie hat keinen Typ; `null` ist ein „positives Nichts“. Ein Klick auf `Abbrechen` in einem Dialog liefert z.B. ein `null` zurück.

Starke Typisierung am Beispiel von Java

Java kennt acht „primitive“ und einige weitere Datentypen. Eine Auswahl:

Datentyp	Inhalt
`boolean`	Logische Wahrheitswerte: `true` / `false`
`char`	Einzelnes Zeichen
`byte`, `short`, `int`, `long`	Ganze Zahlen unterschiedlicher Größe; `byte` ist etwa für Werte zwischen -128 und +127, `long` für positive / negative Wert im Bereich von -2^{64} und $+2^{64}-1$
`float`, `double`	Fließkommazahlen mit unterschiedlicher Genauigkeit: `float` ca. 7 Stellen, `double` ca. 14 Stellen
`String`	Zeichenketten gehören *nicht* zu den „primitiven Datentypen“, sondern sind in Java ein Objektyp, der aus `char` zusammengesetzt ist

Bei einer streng typisierenden Sprache muss bei der Deklaration einer Variablen ihr Datentyp explizit genannt werden. In Java sieht das z.B. so aus:

```
int ganzeZahl = 15000;

float kommaZahl = 3.14;

String text = "Test";

boolean ok = true;

char zeichen= "&";
```

Andere Sprachen haben noch mehr Datentypen, mySQL etwa rund 30, Python 15 etc.

Operatoren / Operanden

Ein **Operator** verbindet zwei Ausdrücke. Die jeweiligen Ausdrücke sind die **Operanden**, z.B.:

`56 + 7`: Operator ist `+`, die Zahlen `56` und `7` sind die Operanden

JavaScript kennt eine Vielzahl verschiedener Operatoren:

- Arithmetische Operatoren (Berechnungen)
- Vergleichsoperatoren (größer, kleiner, gleich)
- Boolesche Operatoren (Logische Ausdrücke)
- Logische Operatoren (Verknüpfung von logischen Ausdrücken)
- Zuweisungsoperator (Wertzuweisungen)

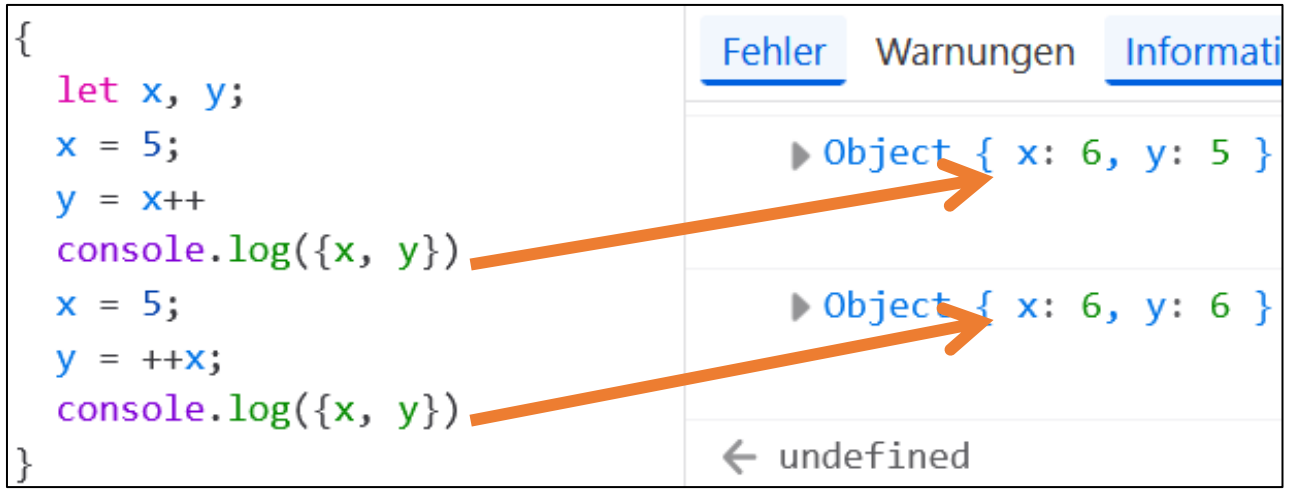
Arithmetische Operatoren

`+ - * /`	Die vier Grundrechenarten: Addition, Subtraktion, Multiplikation, Division.
`%`	Modulo (ganzzahliger Rest einer Division): `7 % 3 = 1`
`**`	Exponent, 5^4 lässt sich etwa mit `5**4` berechnen
`++` `--`	Erhöhung / Verminderung eines Wertes um 1 (Inkrement / Dekrement). Wird sehr oft für Zähler benutzt: `counter++` erhöht den Wert der Variable `counter` um `1`, `counter--` verringert ihn entsprechend

Increment / Decrement

Die Operatoren ++ und -- können sowohl nach dem Operanden (*postfix*) als auch davor (*prefix*) stehen. Je nach Position hat das unterschiedliche Auswirkungen:

- **postfix**: der Wert wird zurückgegeben, anschließend geändert
- **prefix**: der Wert wird geändert, anschließend zurückgegeben



Spielt in der Praxis nur sehr selten eine Rolle … aber kennen sollte man es

Kurzformen für Operatoren

Operationen, bei denen auf beiden Seiten der Zuweisung derselbe Operand auftaucht – also etwa `a = a + 5` oder `b = b * 7` – kommen in der Praxis sehr oft vor. Daher gibt es für diese Operationen Kurzformen

Langform	**Kurzform**
`a = a + 5`	`a += 5`
`a = a - b`	`a -= b`
`a = a * 5`	`a *= 5`
`a = a / 7`	`a /= 7`
`a = a % 3`	`a %= 3`
`a = a**2`	`a **= 2`

Operatorenpräzedenz

Bei einer Anweisung wie `let a = 7 + 5 * 3 - 1;` muss festgelegt werden, in welcher Reihenfolge die Operatoren ausgewertet werden

Dafür gibt es in JavaScript (wie in anderen Sprachen auch) die *Operatorenpräzedenz*: Operatoren mit hoher Präzedenz werden zu Operanden mit niedriger Präzedenz.

Klingt verwirrend? Machen wir’s uns ganz einfach:

- Geklammerte Anweisungen werden zuerst ausgewertet
- Es gilt die „Punkt vor Strich“-Regel: Multiplikation und Division werden vor Addition und Subtraktion ausgewertet
- Operatoren mit gleicher Präzedenz – z.B. `*` und `/` – werden von links nach rechts ausgewertet

Operatorenpräzedenz: Beispiele

```
  7 + 5 * 3 - 1;
= 7 + 15 - 1
= 22 - 1
= 21
```

```
  (7 + 5) * (3 - 1);
= 12 * 2
= 24
```

```
  7 + 5 * (3 - 1);
= 7 + 5 * 2
= 7 + 10
= 17
```

```
  (7 + 5) * 3 - 1;
= 12 * 3 - 1
= 36 - 1
= 35
```

Mit Variablen rechnen

Was mit festen Werten – 5 + 7 = 12 – geht, geht natürlich auch mit Variablen (Variable repräsentieren ja Werte):

```
let zahl1 = 15;
let zahl2 = 30;
let ergAdd = zahl1 + zahl2; // ergAdd hat den Wert 45
let ergSub = zahl1 - zahl2; // ergSub hat den Wert -25
let ergMult = zahl1 * zahl2; // ergMult hat den Wert 450
let ergDiv = zahl1 / zahl2; // ergDiv hat den Wert 0,5
let ergMod = zahl1 % zahl2; // ergMod hat den Wert 15
```

Number + String

Mit dem +-Operator lassen sich nicht nur Zahlen addieren, sondern auch Strings (Zeichenketten) verbinden:

```
let vName = "Peter";
let nName = "Müller";
let fullName = vName + " " + nName; // fullName enthält "Peter Müller"
```

`"test" + 7`	JavaScript interpretiert `7` hier nicht als Zahl, sondern als Text, Ergebnis: `test7`
`"5" + "7"`	Die Zeichenketten werden verknüpft, es entsteht die Zeichenkette `"57"`
`"5" * 3`	JavaScript versucht, den String `"5"` als *Number* zu interpretieren, was geht. Ergebnis: `15`
`"test" * 3`	JavaScript versucht, den String `"test"` als *Number* zu interpretieren, was fehlschlägt. Ergebnis: `NaN` („Not a Number“)

`5 - 4 +"a"`	JavaScript geht von links nach rechts vor:
	`5 – 4 = 1`
	`1 + a = 1a`

String in Number wandeln

Strings sind Zeichenketten; Zeichenketten können auch Zahlen repräsentieren, etwa `"452"`

Bevor man mit so einer Variable rechnen kann, muss der String in eine Zahl umgewandelt werden. Hier gibt es verschiedene Möglichkeiten, z.B.:

```
let a = "123"; // Datentyp: String
a = Number(a); // Datentyp: Number

let b = "456"; // Datentyp: String
b = b * 1;     // Datentyp: Number
```

Sollte die String-Variable nicht als Zahl interpretierbar sein, ist der Wert NaN („Not a Number“), der Datentyp wird aber dennoch auf Number gesetzt

Werte ausgeben

Am einfachsten: `window.alert(wert);` bzw. kurz `alert(wert);`

Werte (Variablen, Konstanten, Literale) lassen sich in der Browser-Konsole mit `console.log(wert);` ausgeben

Bei einer Webseite wird die Anweisung `document.write(wert);` benutzt (es gibt noch andere, bessere Möglichkeiten!)

Mehrere Ausgaben lassen sich kommagetrennt anführen:

```
console.log(wert, 25 + 3, "Test");
```

Innerhalb von Anführungszeichen (= String) können nur die jeweils anderen benutzt werden, also innerhalb von " nur ' und innerhalb von ' nur ":

```
console.log("Er sagte: 'Hallo'");
```

Andernfalls müssen die Zeichen mit einem Backslash \ maskiert werden:

```
console.log("Sie sagte: \"Hallo\"");
```

document.write()

Mit `document.write(String)` wird *String* an der Stelle in das Dokument eingefügt, an der die Anweisung steht.

Die Methode gilt als „deprecated“ (veraltet), zeigt mitunter ein seltsames Verhalten und sollte auf Produktivsystemen zwar *nicht* benutzt werden …

… aber für unsere Zwecke reicht’s erstmal ;-)

Wir werden später bessere und zuverlässigere Methoden kennenlernen, mit denen sich Inhalte in Dokumente einfügen und ändern lassen

Ausgaben: Beispiele

```
document.write(5 + 7);
```

- Es wird das Ergebnis der Berechnung ausgegeben, also 12

```
document.write(myVar);
```

- Es wird der Inhalt der Variable myVar ausgegeben (die Variable muss existieren!)

```
document.write("<p>5 + 7 = ", 5 + 7, "</p>");
```

- Es wird zuerst 5 + 7 = in die Webseite eingefügt, anschließend das Ergebnis der Berechnung (also 12), danach das schließende .

```
document.write("<p>",  a, " + ", b, " = ", a + b, "</p>");
```

- Es wird ein ausgeben, dann der Inhalt der Variable a, dann + , dann der Inhalt der Variable b, dann ein = , dann die Summe von a und b und schließlich das schließende

Template Strings

Mit den „template strings“ lassen sich u.a. Werte in Strings sehr viel leichter einsetzen.

Der String steht in ` ... ` („Backtick“, wird über Shift + Akzenttaste, gefolgt von Leertaste eingegeben).

Die Variable / der Wert steht im String in ${…}

```
let gruss1 = "<p>Hallo, mein Name ist " + vName + "!</p>";
let gruss2 = `<p>Hallo, mein Name ist ${vName}!</p>`;
```

Es können auch Berechnungen durchgeführt werden, etwa:

```
let a = 5;
let b = 7;
let erg = `${a} + ${b} = ${a + b}`;
```

Werte entgegennehmen

Einfachster Weg:
Mit `window.prompt();`
oder kurz `prompt();`

Es erscheint ein Dialogfenster

Der eingegebene Wert kann einer Variablen / Konstanten zugewiesen werden:




```
const name = prompt("Wie heißen Sie?");
```

Obacht! Die übergebenen Werte sind *immer* vom Typ `String`! Soll mit ihnen gerechnet werden, muss zuerst geprüft werden, ob es sich überhaupt um eine Eingabe handelt, die als Zahl interpretiert werden kann; anschließend müssen sie in `Number` umgewandelt werden