Datentyp ermitteln mit typeof-Operator

Operator **typeof** => Rückgabewert = String, der Datentyp angibt

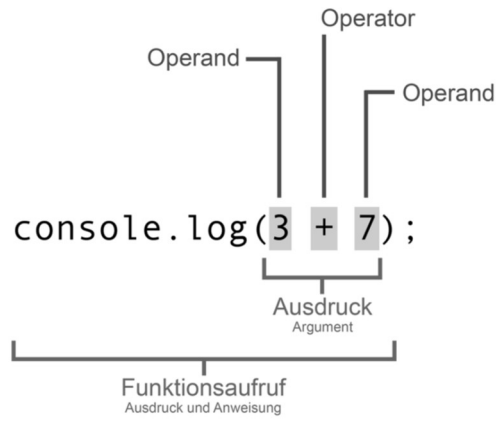
```
"use strict";

console.log(typeof 3789); // => number

console.log(typeof "Java Script"); // => string

console.log(typeof 27.31); // => number
```

Rechnen



Arithmetische Operatoren

Symbol	Operation
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo — Rest einer ganzzahligen Division
**	Potenz (erst ab ECMAScript 2016)

Prioritäten & Klammerung

5 + 8 * 2 = 5 + 16 = 21

(5 + 8) * 2 = 13 * 2 = 26

(12 / (2 +2) + 3) * 2 = (12 / 4 + 3) * 2 = (3 + 3) * 2 = 6 * 2 = 12

Operatoren in Auswertungsreihenfolge

Priorität / Präzedenz	Name	Operator
1	Gruppierungsoperator	Klammern ()
2	typeof-Operator	typeof
3	Multiplikation, Division, Modulo	* / %
4	Addition, Subtraktion	+ -
5	Zuweisungsoperator	=

```
let price = 25;
price = price * 1.16;
console.log(price); // => 29

price *= 1.16;      // das Gleiche wie: price = price * 1.16
price += 10;
```

Priorität / Präzedenz	Name	Operator
1	Gruppierungsoperator	Klammern ()
2	typeof-Operator	typeof
3	Multiplikation, Division, Modulo	* / %
4	Addition, Subtraktion	+ -
5	Zuweisungsoperator	= += -= *= /= %= **=

Zuweisungsoperatoren speichern Wert zurück - funktionieren deswegen nur mit Variablen
25 *= 1.16; // falsch – funktioniert nicht!

Strings

Texte werden in Anführungszeichen geschrieben und bestehen aus einer Aneinanderreihung bzw. Verkettung einzelner Zeichen. Deswegen lautet der Fachbegriff Zeichenkette (engl.: String).

“…“, ‘…‘

Verkettung von Strings

Konkatenations- oder Verkettungsoperator **+**

```
let username = „Jochen“;
console.log("Hi " + username + ". Willkommen zu Hause!");

// => Hi Jochen. Willkommen zu Hause!
```

Implizite Typkonvertierung

Operatoren erwarten als Operanden zwei Zahlen

liegt String als Operand vor führt JS eine implizite Typkonvertierung durch

unabhängig vom Datentyp der beiden Operanden versucht Interpreter, Datentyp zu number umzuwandeln (implizit, da Typkonvertierung automatisch geschieht)

Fachausdruck für Typkonvertierung = coercion

Mathemat. Operatoren (nicht +) verwenden coercion

„5“ * „4“;
„5“ * 4;

Impliziete Typkonvertierung beim Zeichen +

Zeichen + führt bei Operanden vom Typ number keine Typkonvertierung durch

bei Zahlen wirkt als Additionsoperator

wenn auch nur einer der Operanden ein String, konvertiert JS andere Operanden ebenfalls in Strings und wirkt als Verkettungsoperator

```
3 + 4 // => 7

"3" + "4" // => "34"

"3" + 4 // => "34"

3 + "4" // => "34"
```

NaN-Paradoxon

NaN steht für Not a Number

Wert repräsentiert fehlgeschlagene numerische Operation

kann sowohl bei expliziter als auch bei impliziter Typkonvertierung passieren

NaN = Wert vom Typ number – so bleibt Programm im Fehlerfall lauffähig

Number oder nicht Number…

gibt auch Funktionen mit booleschen Rückgabewerten

Funktion isNaN stellt an übergebenes Argument die Frage, ob es sich nicht um eine Zahl handelt

Funktion nutzt implizite Typkonvertierung

ist Argument eine Zahl oder nicht? - true oder false

d.h. Rückgabewert der Funktion ist vom Typ boolean

isNaN(3) ergibt false

isNaN(NaN) ergibt true , denn NaN bedeutet Not a Number

isNaN("Hallo") wird zu NaN konvertiert und ergibt true

isNaN(-42) , isNaN(27) , isNaN(25.342) ergibt false , denn Argumente sind Zahlen

isNaN("17.7") ergibt false

isNaN("dreiundzwanzig") ergibt true

isNaN("3Autos") ergibt true

Math-Objekt

JS verfügt über Reihe mathematischer Funktionen
Werte runden, Zufallszahlen bestimmen, Winkelberechnungen durchführen...
Funktionen werden im Math –Objekt vereint
Zweck dieses Objektes ist, alle passenden Funktionen und Konstanten zu gruppieren (utility object)

Funktion oder Konstante	Zweck	Beispiel
sqrt	Quadratwurzel	Math.sqrt(9) ergibt 3
pow	Potenzieren	Math.pow(2, 5) ergibt 32
cos	Kosinus	Math.cos(0) ergibt 1
PI	π	Math.PI ergibt 3.141592653589793
round	kaufmännisches Runden	Math.round(3.678) ergibt 4
floor	Abrunden	Math.floor(3.678) ergibt 3
ceil	Aufrunden	Math.ceil(3.678) ergibt 4
max	Maximalwert aus beliebig vielen Argumenten	Math.max(3, 7, 2) ergibt 7
min	Minimalwert aus beliebig vielen Argumenten	Math.min(3, 7, 2) ergibt 2
abs	Betrag einer Zahl (ohne Vorzeichen)	Math.abs(-3.678) ergibt 3.678
random	Eine zufällige Zahl gibt eine zufällige Zahl zwischen 0 und 1 zurück, wobei die 1 nie erreicht wird	Math.random()
toFixed	Runden auf beliebig viele Nachkommastellen	3.0.toFixed(2) ergibt 3.00

Relationale Operatoren

Relationale Operatoren ermöglichen das Vergleichen zweier Werte – wie Fragen gelesen

Operator	Operation
<	kleiner
>	größer
<=	kleiner oder gleich
>=	größer oder gleich
==	gleich (versucht Datentypen für Vergleich zu konvertieren)
!=	ungleich (versucht Datentypen für Vergleich zu konvertieren)
===	genau gleich (auch im Typ)
!==	genau ungleich (auch im Typ)

Datentyp Boolean

Datentyp boolean liefert Antwort = passender Rückgabewert

Rückgabewert ist weitere Gemeinsamkeit relationaler Operatoren

Rückgabewerte: **true** oder **false**

Ausdruck, der booleschen Rückgabewert hat (ein boolescher Ausdruck), heißt auch Bedingung

Priorität / Präzedenz	Operator
1	Funktionsaufruf, z. B. prompt(), Klammern ()
2	typeof
3	*, /, %
4	+, -
5	<, > , <=, >=
6	==, !=, ===, !==
7	=, +=, -=, *=, /=, %=

Strings vergleichen

relationale Operatoren wie kleiner oder größer lassen sich auf Strings anwenden

JS führt simplen lexikographischen Vergleich aus

JS vergleicht auch Ziffern Zeichen für Zeichen

Interpreter führt lexikographischen Vergleich durch, wenn beide Operanden vom Typ string

wenn ein Operand eine Zahl, konvertiert Interpreter Datentyp nach number

wandelt sich ein Operand dadurch in NaN ist Ergebnis false

genutzt wird Methode String.charCodeAt() -> liefert Nummer der UTF-16-Zeicheneinheit zurück, die in Zeichenkette an bestimmter Position steht (https://www.fileformat.info/info/charset/UTF-16/list.htm)

Kontrollstrukturen (Steuerstrukturen) steuern Programmabläufe

sind vor allem Verzweigungen und Schleifen

reagieren im Programm auf Bedingungen oder wiederholen Programmschritte

Bedingungen

Programmiersprachen treffen Entscheidungen nach Wenn-dann-Prinzip

Wenn eine Bedingung erfüllt, dann führe eine oder mehrere Anweisungen aus

In JS übernimmt If-Anweisung diese Aufgabe

Anweisung besteht aus:

- Schlüsselwort if
- Bedingung (boolescher Ausdruck) in runden Klammern
- Rumpf in geschweiften Klammern, der Anweisungen enthält

bei if verzweigt sich Programmfluss - gibt zwei mögliche Wege

darum wird if als Verzweigung bezeichnet und der Gruppe der Kontrollstrukturen zugeordnet

If-Anweisung lässt sich erweitern durch **else-Anweisung**

Rumpf der else -Anweisung wird dann ausgeführt, wenn Rumpf der if -Anweisung nicht ausgeführt

else -Anweisung besteht aus:

- Schlüsselwort else
- einem Rumpf mit Anweisungen

else kann nie alleine stehen - ist lediglich Erweiterung für if - daher **if-else-Anweisung**

```
if (Bedingung) {
Anweisung1a;
Anweisung2a;
...
} else {
Anweisung1b;
Anweisung2b;
...
}
```

wenn im if - oder else -Zweig nur eine einzige Anweisung, können wir auf geschweifte Klammern verzichten

wegen Lesbarkeit und Fehleranfälligkeit bei späteren Änderungen Klammern nur dann wegzulassen, wenn kurze Anweisung

```
if (answer === SOLUTION) console.log("42 is correct.");
```

mit **else if()** lassen sich weitere Bedingungen definieren und Verzweigungen verschachteln

Beachten: vom speziellen zum Allgemeinen gehen!

```
if (Bedingung) {
  Anweisung1a;
  Anweisung2a;
  ...
} else if (Bedingung) {
  Anweisung1b;
  Anweisung2b;
  ...
} else {
  Anweisung1b;
  Anweisung2b;
  ...
}
```

Programmierrichtlinien

nach Schlüsselwort if oder else folgt genau ein Leerzeichen

öffnende geschweifte Klammer { des Rumpfes befindet sich in gleicher Zeile wie Schlüsselwort

nach öffnender Klammer des Rumpfes folgt Zeilenumbruch

Anweisungen innerhalb des Rumpfes werden jeweils um 2 Leerzeichen eingerückt

schließende geschweifte Klammer } des Rumpfes in eigene Zeile und linksbündig zum ersten Schlüsselwort setzen

Ausnahme:

enthält Rumpf nur eine Anweisung, dürfen Zeilenumbrüche und geschweifte Klammern entfallen

Logische Operatoren

Oder-Operator || (zwei senkrechte Striche, sog. »Pipes«)

Bedingung_1 || Bedingung_2

Oder-Operator || verbindet beide Bedingungen

wenn mindestens eine Bedingung true zurückgibt, hat Gesamtausdruck im if Rückgabewert true
und if reagiert entsprechend

Wahrheitstabelle des oder-Operators

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

Und-Operator && funktioniert analog

folgt eigener Wahrheitstabelle - Ergebnis einer Und-Verknüpfung ist true, wenn beide Variablen true

Wahrheitstabelle des und-Operators

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

Nicht-Operator wird als Ausrufezeichen ! notiert

wandelt ein true in ein false um - und umgekehrt (invertiert booleschen Wert)

! -Operator hat nur einen Operanden

!true // => false

Wahrheitstabelle des nicht-Operators

a	!a
false	true
true	false

wenn Variable a den Wert false hat, gibt Ausdruck !a Wert true zurück

Logische Operatoren haben geringe Priorität, werden aber noch vor Zuweisung ausgeführt

Nicht-Operator ! als unärer Operator (Operator mit nur einem Operanden) = Ausnahme hat - wie alle unären Operatoren sehr hohe Priorität

Priorität / Präzedenz	Operator
1	Funktionsaufruf, z. B. prompt(), Klammern ()
2	!, typeof
3	*, /, %
4	+, -
5	<, > , <=, >=
6	==, !=, ===, !==
7	&&
8	||
9	=, +=, -=, *=, /=, %=