# Skriptum Kommunikationstechnik

**Teil 2 Datenformate**

DI(FH) Andreas Pötscher, HTL Litec

1.9.2025

## Inhaltsverzeichnis





| Datum | Beschreibung der durchgeführten Änderung | Autor |
|---|---|---|
| 09.04.2025 | V1.0 . . . 1.Version Basierend auf FI1 bis FI3 von Gebhard Klinkan | Andreas Pötscher, HTL Linz–Paul-Hahn-Straße (LiTec) |

## Einführung JavaScript

Einfachen Daten werden in IT Systemen als primitive Datentypen dargestellt. Diese können je nach Datentyp verschiedene Werte enthalten. In dieser Unterlage wird dafür die Programmiersprache JavaScript verwendet. Auf dieser basiert auch Node-Red, das für die Übungsbeispiele verwendet wird. In JS (JavaScript) gibt es folgende primitive Datentypen:

- **Number**: Für numerische Werte, sowohl Ganzzahlen als auch Gleitkommazahlen.
- **String**: Für Zeichenketten, also Text.

- **Boolean**: Für Wahrheitswerte `true` oder `false`.
- **Null**: Wert, der bewusst als Abwesenheit von einem Wert verwendet wird.

Die Syntax von JS basiert auf der C Syntax.

### Dynamische Typisierung

Im großen Unterschied zu C verwendet JavaScript keine strenge Typisierung sondern eine dynamische Typisierung, das sogenannte Duck Typing. Duck Typing ist ein Konzept in JavaScript, bei dem der Datentyp eines Objekts oder einer Variable nicht aufgrund der Initialisierung, sondern aufgrund seines Verhaltens oder seiner Eigenschaften bestimmt wird. Dies bedeutet, dass JavaScript die Typüberprüfung zur Laufzeit durchführt, anstatt zur Entwicklungszeit.

Beispiel:

```
let i = 10;
console.log(typeof(i)); //number
```

Das die Variable i vom Typ number ist wird erst dynamisch durch die Zuweisung des Wertes 10 festgelegt.

Beispiel:

```
let i = "10"
console.log(typeof(i)); //string
```

Hier wird die Variable i durch Zuweisung eines Textes automatisch zu einem String.

Diese Verhalten ist an und für sich sehr praktisch beinhaltet aber auch einige Fallstricke. Javascript wandelt (auch casting genannt) wenn möglich die Variablen automatisch auf Datentypen, dass sich die Operation ausführen lassen.

Beispiel:

```
let a = 1;   //number
let b = "1"  //string
console.log(b+a); //Ausgabe 11
```

Hier wird a automatisch auf einen String gecastet, da der + Operator für Strings implementiert ist. Der + Operator verkettet Strings miteinander.

Beispiel:

```
let a = 1;
let b = "1"
console.log(b-a); // Ausgabe 0
```

Der - Operator ist für Strings nicht implementiert. Deshalb wird b automatisch auf den Typ number gecastet.

Um hier Fehler zu vermeiden, können die Variablen aktiv gecastet werden.

Beispiel:

```
let a = 1;
let b = "1"
console.log(a+Number(b)); // Ausgabe 2
```

### Kontrollstrukturen in JS

Bei den Kontrollstrukturen gibt es im weitesten Sinne die gleichen wie in C/C++.

- Verzweigungen (`if`,`switch`)
- Schleifen (`for`, `while`)

Die Vergleichsoperatoren wie `&&`, `||`, `!`, usw funktionieren wie in C. Aus der dynamischen Typisierung ergibt sich dennoch ein wichtiger Unterschied. Es gibt Vergleichsoperatoren die nur den Inhalt der Variable vergleichen und Vergleichsoperatoren die auch den Datentyp vergleichen.

| Operator | Was wird verglichen | Beispiel | |
|---|---|---|---|
| `==` | Wert | `10 == "10"` | `//true` |
| `===` | Wert und Datentyp | `10 === "10"` | `//false` |
| `!=` | Wert | `10 != "10"` | `//false` |
| `!==` | Wert und Datentyp | `10 !== "10"` | `//true` |

Beispiel:

```
let x = 10;
let y = "10";
console.log(x == y);  //true
console.log(x === y); //false
```

## Darstellung von komplexeren Datenstrukturen in Software

In diesem Skriptum werden zwei komplexe Datenstrukturen in JavaScript beschrieben. Arrays und Objekte.

### Arrays in JS

Arrays in JavaScript sind spezielle Datentypen, die verwendet werden, um eine geordnete Liste von Werten zu speichern. Arrays können Werte unterschiedlicher Datentypen (einschließlich anderer Arrays) enthalten und ermöglichen den Zugriff auf Elemente anhand ihres Index. Theoretisch handelt es sich bei JS Arrays nicht klassische Arrays sondern um Listen. Deshalb gibt es auch Funktion mit denen Elemente hinzugefügt und entfernt werden können.

**Deklaration von Arrays:** Um ein Array zu erstellen, können Sie die folgenden beiden Methoden verwenden:

```
// Methode 1: Verwenden Sie die Array-Literal-Syntax
const fruits = ["Apfel", "Banane", "Kirsche"];

// Methode 2: Verwenden Sie den Array-Konstruktor
const colors = new Array("Rot", "Grün", "Blau");
```

**Zugriff auf Array-Elemente:** Sie können auf die Elemente eines Arrays mithilfe ihres Index zugreifen. Beachten Sie, dass die Indexierung in JavaScript bei 0 beginnt:

```
console.log(fruits[0]); // Gibt "Apfel" aus
console.log(colors[2]); // Gibt "Blau" aus
```

**Länge eines Arrays:** Sie können die Länge eines Arrays mithilfe der `length`-Eigenschaft ermitteln:

```
console.log(fruits.length); // Gibt 3 aus
```

**Hinzufügen von Elementen:** Sie können ein Element am Ende eines Arrays hinzufügen, indem Sie die `push()`-Methode verwenden:

```
fruits.push("Erdbeere");
console.log(fruits); // Gibt ["Apfel", "Orange", "Kirsche", "Erdbeere"] aus
```

**Entfernen von Elementen:** Am Ende des Array kann ein Element mit `pop()` Funktion entfernt werden.

```
fruits.pop();
console.log(fruits); // Gibt ["Apfel", "Orange", "Kirsche"] aus
```

### Klassen in JS

## Darstellung von Daten in Datenaustauschformaten

Strukturierte Darstellung von Daten. CSV / JSON / XML

### Java Script Object Notation (JSON)

https://datatracker.ietf.org/doc/html/rfc8259

### eXtensible Markup Language (XML)

https://datatracker.ietf.org/doc/html/rfc3470

### Comma Separated Value (CSV)

https://datatracker.ietf.org/doc/html/rfc4180