3.1 – JSA: Primitive Datentypen und Typumwandlung

Hintergrund und Zielsetzung

JavaScript kennt verschiedene primitive Datentypen wie `string`, `number`, `boolean`, `undefined`, `null`, `bigint` und `symbol`. Sie bilden die Grundlage für alle Operationen mit Werten, die nicht referenziert oder verändert werden sollen. Gleichzeitig stellt JavaScript Wrapper-Objekte wie `Number`, `String` oder `Boolean` bereit, die primitive Werte objektartig nutzbar machen.

Ziel dieses Kapitels ist es, die Unterschiede zwischen primitiven Werten und Objektvarianten, die Besonderheiten bei Typumwandlungen und das Verhalten bei Methodenaufrufen auf primitiven Typen vollständig zu verstehen. Dabei werden auch Spezialwerte wie `NaN`, `Infinity` oder Besonderheiten wie Autoboxing, `typeof` und Falsy-Werte behandelt.

Alle Beispiele sind für Live-Demonstrationen im Unterricht geeignet. Ergänzt wird jedes Thema durch MDN-Links zur offiziellen Referenz.

---

1. Übersicht der primitiven Datentypen

Typ	Beispiel	Beschreibung
`string`	"Hallo"	Zeichenkette
`number`	42, 3.14	Gleitkommazahl (64-Bit IEEE 754)
`boolean`	true, false	Wahrheitswert
`null`	null	absichtlich leerer Wert
`undefined`	undefined	nicht initialisierter Wert
`bigint`	123456789123456789n	Ganzzahlen > 2^53-1
`symbol`	Symbol("id")	eindeutiger, nicht kollidierender Bezeichner

---

2. Autoboxing – Methoden auf primitiven Werten

Auch primitive Werte verhalten sich oft wie Objekte, obwohl sie das technisch nicht sind. Hintergrund ist der Mechanismus des **Autoboxing**.

Was passiert?

Wenn auf einem primitiven Wert eine Methode aufgerufen wird, z. B.:

```
let str = "hallo";
console.log(str.toUpperCase());
```

... erzeugt JavaScript intern **temporär** ein entsprechendes Objekt (`new String(str)`), führt die Methode aus und verwirft das Objekt direkt danach wieder.

Dieser Vorgang gilt auch für Zahlen und Booleans:

```
let num = 42;
console.log(num.toFixed(2)); // "42.00"

let flag = true;
console.log(flag.toString()); // "true"
```

> MDN: Autoboxing-Konzept erklärt

Wichtig: Diese temporären Objekte sind **nicht identisch** mit dauerhaft erzeugten Wrapper-Objekten (`new Number(42)` etc.). Letztere führen zu unerwartetem Verhalten und sollten vermieden werden.

---

3. `number`: Zahlen in JavaScript

```
let a = 42;
let b = 3.14;
let c = -0;
let d = Infinity;
let e = NaN;
```

Konstruktion

```
let x = new Number(42);     // Objekt
let y = Number("42");       // primitive Zahl
```

> MDN: Number()

Eigenschaften

```
console.log(Number.MAX_VALUE);          // größte darstellbare Zahl
console.log(Number.MIN_VALUE);          // kleinste positive Zahl
console.log(Number.NaN);                // Not a Number
console.log(Number.POSITIVE_INFINITY);
console.log(Number.isNaN("abc"));       // false (da string)
```

> MDN: Number Properties

Methoden

```
let n = 1234.567;
console.log(n.toFixed(2));          // "1234.57"
console.log(n.toExponential(1));  // "1.2e+3"
console.log(n.toPrecision(4));    // "1235"
console.log(n.toLocaleString("de-DE")); // "1.234,567"
```

> MDN: Number.prototype.toLocaleString()

---

4. `boolean`: Wahrheitswerte

```
let x = true;
let y = false;
```

Konstruktion

```
let a = new Boolean(true);   // Objekt
let b = Boolean("text");    // true
```

> MDN: Boolean()

Verhalten

```
console.log(Boolean(0));     // false
console.log(Boolean(""));    // false
console.log(Boolean([]));    // true
```

> Achtung: Ein `Boolean`-Objekt ist immer truthy, auch wenn `false` gespeichert ist:

```
let obj = new Boolean(false);
if (obj) console.log("Wird ausgeführt"); // true!
```

---

5. `string`: Zeichenketten

```
let s = "Text";
let s2 = new String("Text"); // Objekt
```

> MDN: String

Eigenschaften

```
console.log(s.length);         // 4
```

Methoden (Auswahl)

```
s.toUpperCase();               // "TEXT"
s.toLowerCase();               // "text"
s.charAt(1);                   // "e"
s.slice(0, 2);                 // "Te"
s.replace("e", "a");          // "Tast"
s.includes("xt");             // true
s.padStart(6, "-");            // "--Text"
s.trim();                      // entfernt Leerzeichen
s.localeCompare("Text");      // 0
```

> MDN: String Methods

---

6. Weitere primitive Typen

`undefined`

```
let x;
console.log(x); // undefined
```

`null`

```
let y = null;
console.log(typeof y); // "object" (JS-Bug)
```

`bigint`

```
let large = 1234567890123456789012345678901234567890n;
```

> MDN: BigInt

`symbol`

```
let id = Symbol("userId");
```

> MDN: Symbol

---

7. Typumwandlung (Type Conversion)

Explizit

```
String(42);          // "42"
Boolean(0);          // false
Number("3.14");      // 3.14
parseInt("42abc");   // 42
parseFloat("3.14");  // 3.14
```

> MDN: Type Conversion

Implizit

```
"5" + 1      // "51"
"5" - 1      // 4
true + 2     // 3
null + 1     // 1
undefined + 1 // NaN
```

Falsy-Werte

- `false`, `0`, `-0`, `""`, `null`, `undefined`, `NaN`

```
if ("") console.log("Wird NICHT ausgeführt");
```

---

8. Typprüfung

```
typeof "abc";        // "string"
typeof 42;           // "number"
typeof null;         // "object" (JS-Bug)
typeof undefined;    // "undefined"
typeof [];           // "object"

[] instanceof Array; // true
```

> MDN: typeof

---

9. Vergleich mit Python

Konzept	JavaScript	Python
Primitive	string, number, boolean, null	str, int, float, bool, None
Wrapper	`String`, `Number`	implizit
Autoboxing	Ja	Nein
Type Coercion	Automatisch	Nur explizit
Typprüfung	`typeof`, `instanceof`	`type()`, `isinstance()`
Falsy-Werte	viele	`False`, `0`, `None`, `""`

---

10. Übungsaufgaben

Aufgabe 1: Typ erkennen

```
typeof 42
typeof "42"
typeof new Number(42)
typeof true
typeof Boolean(false)
typeof undefined
```

Aufgabe 2: Konvertieren

Wandle folgende Werte explizit um:

- Zahl `0` in String
- String "123abc" in Integer
- `null` in Boolean
- String "12.75" in Float

Aufgabe 3: Autoboxing verstehen

```
let text = "Hallo";
console.log(text.length); // Warum funktioniert das?
```

Aufgabe 4: Typ-Coercion analysieren

```
"4" + 2
"4" - 2
true + 3
false + "7"
null + 1
undefined + 1
```

Aufgabe 5: Vergleich zu Python

```
Boolean(0)
String(true)
Number("12.5")
```

Aufgabe 6: Falsy-Filter

```
function filterTruthy(arr) {
  return arr.filter(Boolean);
}

filterTruthy([0, "", null, "ok", undefined, 5]); // → ["ok", 5]
```

---

11. Micro-Projekt: Eingabekonverter

Ziel

Ein Konverter für Texteingaben, der automatisch erkennt, ob es sich um Zahlen, Booleans, leere Strings oder Texte handelt und entsprechend konvertiert.

Spezifikation

- Eingabe: Array aus Strings, z. B. `['42', 'true', '', 'Text']`
- Ausgabe: Array mit typisierten Werten
  - "42" → `number`
  - "true" → `boolean`
  - "" → `null`
  - alles andere bleibt `string`

Beispiel

```
function convertInputs(arr) {
  return arr.map(val => {
    if (val === "") return null;
    if (val === "true") return true;
    if (val === "false") return false;
    if (!isNaN(val)) return parseFloat(val);
    return val;
  });
}

console.log(convertInputs(["42", "true", "", "Hallo"]));
// → [42, true, null, "Hallo"]
```

Erweiterungsideen

- Typ-Validierung anhand regulärer Ausdrücke
- Eingabe via Formularfeld und Live-Ausgabe
- Rückwärtskonvertierung für API-Kommunikation