JavaScript Assignment - Woche 1

Vorbereitung auf React/TypeScript Kurs

> **Bearbeitungszeit:** 6-8 Stunden
> **Abgabefrist:** Vor Kursbeginn
> **Bewertung:** Selbstkorrektur mit bereitgestellten Lösungen
> **Lernziel:** Alle Grundlagen (Kapitel 1-2) praktisch anwenden

---

📋 Allgemeine Anforderungen

Code-Qualität

- ✅ Aussagekräftige Variablennamen (camelCase)
- ✅ Kommentare für komplexe Logik
- ✅ `const` bevorzugen, `let` nur wenn nötig, **NIEMALS** `var`
- ✅ `===` statt `==` verwenden
- ✅ Fehlerbehandlung wo sinnvoll (try-catch)

Abgabeformat

Erstelle eine Datei `assignment-week1.js` mit allen Lösungen.

🎯 Teil 1: Grundlagen & Variablen (15 Punkte)

Aufgabe 1.1: Benutzerprofil (5 Punkte)

Erstelle Variablen für ein Benutzerprofil und gib sie formatiert aus:

Anforderungen:

- Nutze die richtigen Datentypen (string, number, boolean)
- Verwende `const` für unveränderliche Werte
- Nutze Template Literals für die Ausgabe

Beispiel-Output:

```
=== Benutzerprofil ===
Name: Max Mustermann
Alter: 28 Jahre
Premium-Mitglied: Ja
Kontostand: 1.234,56 €
Mitglied seit: 2020
```

Zusatzaufgabe (+2 Punkte):

- Berechne, wie viele Jahre der User schon Mitglied ist
- Prüfe, ob der User volljährig ist
- Formatiere den Kontostand mit 2 Nachkommastellen

Aufgabe 1.2: Datentypen erkennen (5 Punkte)

Erstelle eine Funktion `analysiereWert(wert)` die:

- Den Typ des Werts ermittelt (`typeof`)
- Prüft ob der Wert "truthy" oder "falsy" ist
- Eine aussagekräftige Beschreibung zurückgibt

Test-Werte:

```javascript
analysiereWert(42);
analysiereWert("Hallo");
analysiereWert(null);
analysiereWert(undefined);
analysiereWert([1, 2, 3]);
analysiereWert({ name: "Max" });
```

Beispiel-Output:

```
Wert: 42
Typ: number
Truthy: ja
Beschreibung: Eine Zahl
```

Aufgabe 1.3: Operatoren-Quiz (5 Punkte)

Berechne folgende Ausdrücke **ohne sie auszuführen** (schreibe deine Vermutung als Kommentar). Dann lass den Code laufen und prüfe deine Vermutungen:

```javascript
// Was kommt raus?
console.log(5 + "5");       // Deine Vermutung:
console.log("10" - 5);      // Deine Vermutung:
console.log(true + true);   // Deine Vermutung:
console.log(null ?? 10);    // Deine Vermutung:
console.log(0 || 100);      // Deine Vermutung:
console.log("" && "Hallo"); // Deine Vermutung:
```

Zusatzaufgabe (+3 Punkte): Erkläre in 1-2 Sätzen **WARUM** jedes Ergebnis so ist.

🔀 Teil 2: Kontrollstrukturen (25 Punkte)

Aufgabe 2.1: Notenverwaltung (8 Punkte)

Schreibe ein Programm, das Noten in Buchstaben umwandelt:

Anforderungen:

- Funktion `berechneNote(punkte)`
- Punkte: 0-100
- Noten: A (90-100), B (80-89), C (70-79), D (60-69), F (<60)
- Fehlerbehandlung für ungültige Eingaben

Erweitert (+3 Punkte):

- Berechne den Durchschnitt mehrerer Noten
- Prüfe, ob der Durchschnitt zum Bestehen reicht (≥60)
- Finde die beste und schlechteste Note

Test-Daten:

```javascript
const noten = [85, 92, 78, 65, 88, 95, 72];
```

Beispiel-Output:

Noten: [85, 92, 78, 65, 88, 95, 72]
Durchschnitt: 82.14 Punkte (Note: B)
Status: Bestanden ✓
Beste Note: 95 (A)
Schlechteste Note: 65 (D)

Aufgabe 2.2: Login-System (7 Punkte)

Simuliere ein einfaches Login-System:

Anforderungen:

- Funktion `login(username, password, maxVersuche)`
- Maximale Anzahl Versuche (z.B. 3)
- Nach 3 Fehlversuchen: Account gesperrt
- Erfolgreicher Login gibt Begrüßung aus

Korrekte Daten für Test:

```javascript
const korrekterUser = "admin";
const korrektesPW = "geheim123";
```

Simuliere Versuche:

```javascript
// Versuch 1: Falscher Username
login("user", "test123");

// Versuch 2: Falsches Passwort
login("admin", "falsch");

// Versuch 3: Korrekt
login("admin", "geheim123");
```

Beispiel-Output:

Versuch 1/3: ❌ Falsche Anmeldedaten
Versuch 2/3: ❌ Falsche Anmeldedaten
Versuch 3/3: ✅ Willkommen, admin!

Aufgabe 2.3: Einkaufswagen (10 Punkte)

Erstelle ein Einkaufswagen-System:

Anforderungen:

- Array von Produkten mit Preis und Anzahl
- Berechne Gesamtsumme
- Wende Rabatt-Regeln an:
  - 10% ab 100€
  - 15% ab 200€
  - 20% ab 300€
- Prüfe auf kostenlose Lieferung (ab 50€)
- Verwende `switch` für Lieferoptionen

Produkte:

```javascript
const warenkorb = [
  { name: "Laptop", preis: 899.99, anzahl: 1 },
  { name: "Maus", preis: 25.50, anzahl: 2 },
  { name: "Tastatur", preis: 79.90, anzahl: 1 },
  { name: "Monitor", preis: 299.00, anzahl: 1 }
];
```

Lieferoptionen:

```javascript
const lieferung = "standard"; // "standard", "express", "abholung"
```

Beispiel-Output:

```
=== EINKAUFSWAGEN ===
Laptop x1: 899,99 €
Maus x2: 51,00 €
Tastatur x1: 79,90 €
Monitor x1: 299,00 €
--------------------
Zwischensumme: 1.329,89 €
Rabatt (20%): -265,98 €
Lieferung (Standard): KOSTENLOS ✓
--------------------
GESAMT: 1.063,91 €
```

🔄 Teil 3: Schleifen & Arrays (20 Punkte)

Aufgabe 3.1: Zahlenreihen (6 Punkte)

Erstelle Funktionen für verschiedene Zahlenreihen:

a) Gerade Zahlen (2 Punkte)

```javascript
function geradeZahlen(start, ende) {
  // Gib alle geraden Zahlen zwischen start und ende aus
}

geradeZahlen(1, 20); // 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
```

b) Fibonacci (4 Punkte)

```javascript
function fibonacci(anzahl) {
  // Erste n Fibonacci-Zahlen: 0, 1, 1, 2, 3, 5, 8, 13...
  // Jede Zahl ist Summe der beiden vorherigen
}

fibonacci(10); // [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Aufgabe 3.2: Primzahlen-Finder (8 Punkte)

Schreibe ein Programm das Primzahlen findet:

Anforderungen:

- Funktion `istPrimzahl(zahl)` - prüft ob Zahl prim ist
- Funktion `findePrimzahlen(start, ende)` - findet alle Primzahlen in Bereich
- Optimiere mit `break` (früher Ausstieg wenn nicht prim)

Bonus (+3 Punkte):

- Zähle wie viele Primzahlen gefunden wurden
- Berechne deren Summe
- Finde die größte Primzahl im Bereich

Test:

```javascript
findePrimzahlen(1, 50);
```

Beispiel-Output:

```
Primzahlen von 1 bis 50:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Anzahl: 15
Summe: 328
Größte: 47
```

Aufgabe 3.3: Studentenverwaltung (6 Punkte)

Verwalte eine Liste von Studenten:

Daten:

```javascript
const studenten = [
  { id: 1, name: "Anna Schmidt", note: 1.7, anwesend: true },
  { id: 2, name: "Max Müller", note: 2.3, anwesend: false },
  { id: 3, name: "Lisa Weber", note: 1.0, anwesend: true },
  { id: 4, name: "Tom Becker", note: 3.7, anwesend: true },
  { id: 5, name: "Sarah Klein", note: 2.0, anwesend: false }
];
```

Aufgaben:

1. Finde alle anwesenden Studenten
2. Berechne Durchschnittsnote aller Studenten
3. Finde den besten Studenten (niedrigste Note)
4. Zähle wie viele Studenten bestanden haben (Note ≤ 4.0)
5. Erstelle eine Liste nur mit Namen (alphabetisch sortiert)

Beispiel-Output:

```
=== ANWESENHEIT ===
Anwesend: 3 von 5 Studenten
- Anna Schmidt
- Lisa Weber
- Tom Becker

=== NOTENSPIEGEL ===
Durchschnittsnote: 2.14
Beste Note: Lisa Weber (1.0)
Bestandenquote: 100%

=== NAMENSLISTE ===
1. Anna Schmidt
2. Lisa Weber
3. Max Müller
4. Sarah Klein
5. Tom Becker
```

🛡️ Teil 4: Fehlerbehandlung (15 Punkte)

Aufgabe 4.1: Sichere Rechner-Funktion (7 Punkte)

Erstelle eine Rechner-Funktion mit Fehlerbehandlung:

Anforderungen:

- Funktionen für +, -, *, /
- Fehlerbehandlung für:
  - Division durch 0
  - Ungültige Operatoren
  - Keine Zahlen übergeben
- Nutze `try-catch-finally`

Beispiel:

```javascript
function rechner(a, b, operator) {
  // Deine Implementierung
}

rechner(10, 5, "+");   // 15
rechner(10, 0, "/");   // Fehler: Division durch 0
rechner("a", 5, "+");  // Fehler: Ungültige Eingabe
rechner(10, 5, "%");   // Fehler: Unbekannter Operator
```

Aufgabe 4.2: JSON Parser (8 Punkte)

Erstelle ein sicheres JSON-Parsing-Tool:

Anforderungen:

- Funktion `parseUserData(jsonString)`
- Validiere das JSON-Format
- Prüfe erforderliche Felder (name, email, age)
- Validiere Datentypen
- Gib aussagekräftige Fehlermeldungen

Test-Daten:

```javascript
const validJSON = '{"name":"Max","email":"max@test.de","age":25}';
const invalidJSON = '{name:"Max"}'; // Ungültiges JSON
const missingField = '{"name":"Max","email":"max@test.de"}'; // age fehlt
const wrongType = '{"name":"Max","email":"max@test.de","age":"25"}'; // age ist string
```

Beispiel-Output:

```
✅ Valide Benutzerdaten:
Name: Max
Email: max@test.de
Alter: 25 Jahre

❌ Fehler: Ungültiges JSON-Format
❌ Fehler: Pflichtfeld 'age' fehlt
❌ Fehler: 'age' muss eine Zahl sein
```

🎨 Teil 5: Kombinierte Challenge (25 Punkte)

Aufgabe 5: Todo-List Manager

Erstelle eine vollständige Todo-List Verwaltung (Vorbereitung auf React!):

Anforderungen:

1. Datenstruktur (5 Punkte)

```javascript
const todos = [
  { id: 1, titel: "JavaScript lernen", erledigt: false, priorität: "hoch", erstellt: "2024-11-01" },
  { id: 2, titel: "React Kurs vorbereiten", erledigt: false, priorität: "hoch", erstellt: "2024-11-02" },
  { id: 3, titel: "Einkaufen", erledigt: true, priorität: "niedrig", erstellt: "2024-10-30" }
];
```

2. Funktionen (20 Punkte)

a) todoHinzufügen(titel, priorität) (3 Punkte)

- Neues Todo erstellen
- Automatische ID vergeben
- Aktuelles Datum setzen

b) todoErledigen(id) (2 Punkte)

- Todo als erledigt markieren
- Fehlerbehandlung wenn ID nicht existiert

c) todoLöschen(id) (2 Punkte)

- Todo entfernen
- Warnung ausgeben

d) todosFiltern(filter) (4 Punkte)

- Filter: "alle", "offen", "erledigt"
- Sortierung nach Priorität

e) todosNachPriorität() (3 Punkte)

- Gruppiere Todos: hoch, mittel, niedrig
- Zähle pro Gruppe

f) statistik() (3 Punkte)

- Gesamtanzahl
- Erledigt / Offen
- Prozentsatz erledigt
- Ältestes Todo

g) exportieren() (3 Punkte)

- Erstelle formatierten Text-Output
- Oder JSON-String

Beispiel-Output:

```
=== TODO-LIST MANAGER ===

📝 Alle Todos (3):
[HOCH] ⬜ JavaScript lernen (seit 10 Tagen)
[HOCH] ⬜ React Kurs vorbereiten (seit 9 Tagen)
[NIEDRIG] ✅ Einkaufen (seit 12 Tagen)

📊 Statistik:
Gesamt: 3 Todos
Erledigt: 1 (33%)
Offen: 2 (67%)
Ältestes Todo: Einkaufen (12 Tage)

🎯 Nach Priorität:
HOCH: 2 Todos (0 erledigt)
MITTEL: 0 Todos
NIEDRIG: 1 Todo (1 erledigt)

🔍 Filter "offen":
[HOCH] JavaScript lernen
[HOCH] React Kurs vorbereiten
```

🌟 Bonus-Challenge: TypeScript Preview (10 Extra-Punkte)

Wandle deine Todo-List in TypeScript um:

Aufgaben:

1. Definiere ein `Todo` Interface
2. Definiere Typen für Priorität (`'hoch' | 'mittel' | 'niedrig'`)
3. Definiere Typen für Filter
4. Füge Typen zu allen Funktionen hinzu

Beispiel:

```typescript
```

```typescript
interface Todo {
  id: number;
  titel: string;
  erledigt: boolean;
  priorität: Priorität;
  erstellt: string;
}

type Priorität = 'hoch' | 'mittel' | 'niedrig';
type Filter = 'alle' | 'offen' | 'erledigt';

function todoHinzufügen(titel: string, priorität: Priorität): Todo {
  // Implementierung
}
```

📊 Bewertungskriterien

Punkte-Verteilung:

- Teil 1 (Grundlagen): 15 Punkte
- Teil 2 (Kontrollstrukturen): 25 Punkte
- Teil 3 (Schleifen): 20 Punkte
- Teil 4 (Fehlerbehandlung): 15 Punkte
- Teil 5 (Challenge): 25 Punkte
- **Gesamt: 100 Punkte**
- **Bonus (TypeScript): +10 Punkte**

Bewertung:

- 90-110 Punkte: ⭐ ⭐ ⭐ ⭐ ⭐ Hervorragend
- 75-89 Punkte: ⭐ ⭐ ⭐ ⭐ Sehr gut
- 60-74 Punkte: ⭐ ⭐ ⭐ Gut
- 50-59 Punkte: ⭐ ⭐ Befriedigend
- <50 Punkte: ⭐ Nochmal üben!

💡 Tipps zur Bearbeitung

Vorgehen:

Zeitplanung:

- Teil 1: ~1 Stunde
- Teil 2: ~2 Stunden
- Teil 3: ~1,5 Stunden
- Teil 4: ~1 Stunde
- Teil 5: ~2 Stunden
- **Gesamt: 7-8 Stunden** (auf 2-3 Tage verteilen!)

Erlaubte Hilfsmittel:

- ✅ MDN Web Docs (developer.mozilla.org)
- ✅ Deine bisherigen Lernmaterialien
- ✅ Console.log() zum Debuggen
- ❌ Keine fertigen Lösungen kopieren!
- ❌ Keine KI (ChatGPT, Claude) für komplette Lösungen

📤 Abgabe & Selbstkorrektur

Format:

```
assignment-week1/
├── teil1.js
├── teil2.js
├── teil3.js
├── teil4.js
├── teil5.js
└── bonus-typescript.ts (optional)
```

Selbstkorrektur:

Nach Abschluss werden Musterlösungen bereitgestellt.

Vergleiche deine Lösung und notiere:

- Was hast du anders gemacht?
- Was hast du gelernt?
- Was war schwierig?

🎯 Lernziele Check

Nach diesem Assignment solltest du können:

- ✅ Mit allen Datentypen sicher umgehen
- ✅ Bedingungen und Schleifen kombinieren
- ✅ Arrays effektiv durchlaufen und verarbeiten
- ✅ Fehler sauber behandeln
- ✅ Funktionen sinnvoll strukturieren
- ✅ Code lesbar und wartbar schreiben
- ✅ **Bereit sein für React!**

Viel Erfolg! 🚀

Bei Fragen oder Problemen: Notiere sie und kläre sie im Kurs.