Lorenz Glißmann
Alrun Naake
Georg-August-Universität Göttingen
Institut für Informatik

Übung 01: JavaScript

Statt weniger großer Aufgaben gibt es in dieser Übung viele kleine Aufgaben.

Aufgabe 1 (10 min)

Installiere Node.js und npm (Node Package Manager) auf deinem Rechner.

Mit Admin-Rechten

```shell
# Alte Paketversion von Ubuntu entfernen
sudo apt-get remove nodejs
# Installiere die neueste Version von Node.js
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Ohne Admin-Rechte

Die original Anleitung findest du hier: https://github.com/nvm-sh/nvm?tab=readme-ov-file#installing-and-updating

```shell
# Installiere nvm (Node Version Manager)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
source ~/.config/nvm/nvm.sh
nvm install node
```

Aufgabe 1 - Bun

Installiere Bun (https://bun.sh/):

```shell
curl -fsSL https://bun.sh/install | bash
```

Aufgabe 1 - Einfache JavaScript-Programme

1. Schreibe ein JavaScript-Programm, das den Text „Hello World“ in der Konsole ausgibt.
2. Schreibe ein JavaScript-Programm, dass die ersten 10 Quadratzahlen in einer `for`-Schleife berechnet und in der Konsole ausgibt.
3. Schreibe ein JavaScript-Programm, dass die ersten 10 Quadratzahlen in einer `for`-Schleife berechnet und ausgibt, indem sie die **ersten `n` ungeraden Zahlen aufsummiert**.
4. Schreibe eine JavaScript-**Funktion**, die die ersten `n` Quadratzahlen berechnet und als Array zurück gibt.
5. Schreibe eine JavaScript-Funktion, in der ein Array der ersten `n` Zahlen erstellt wird (siehe Folien). Berechne folgende Werte aus dem Array:
   1. die Summe der Zahlen mit einer `for-of`-Schleife
   2. die Summe der Zahlen mit `.forEach`
   3. die Summe der Zahlen mit einer `.reduce`-Funktion (ohne eine `let`-Variable)
   4. die ersten `n` Quadratzahlen mit `.map`

Aufgabe 2 - Einfache HTML-Seite generieren lassen

Schreibe ein TypeScript-Programm, dass mithilfe von Template-Strings eine HTML-Seite für einen Steckbrief (Vorstellung einer Person) erstellt. Erstelle dafür zunächst ein *einfaches* Datenmodell[1] für eine Person und erstelle dann eine Funktion, die eine HTML-Seite (als String) für diese Person erstellt.

Du kannst TypeScript-Dateien mit `bun` ausführen. Erstelle dafür eine Datei (z.B. `index.ts`) und führe sie mit dem Befehl `bun index.ts` aus. Du kannst auch den Befehl `bun run index.ts` verwenden, um die Datei auszuführen.

Erstelle anschließend eine Datenstruktur, die mehrere Personen beinhaltet. Erstelle jeweils eine HTML-Seite für jede Person und speichere sie in einer Datei `<name>.html` ab. Du kannst dafür die `fs`-Bibliothek, die node und bun mitbringen:

```ts
import fs from "fs";
const html = `<!DOCTYPE html> ...`;
fs.writeFileSync("index.html", html);
```

> ⚠ Vorsicht: Dieser Code ist synchron, d.h. er blockiert den Event-Loop, bis die Datei geschrieben ist. In der Praxis solltest du asynchrone Funktionen verwenden, um die Performance zu verbessern.

Aufgabe 3 - Asynchrones JavaScript/TypeScript

Das folgende Code-Snippet verzögert die Ausgabe von „Hello World!“ um 1 Sekunde.

```js
const delay(time) => new Promise(resolve => setTimeout(resolve, time));
delay(1000).then(() => console.log("Hello World!"));
```

1. Schreibe eine Funktion `squareVoid(n: number): void`, die eine Zahl als Parameter entgegennimmt, die Zahl sofort ausgibt und mit 1 Sekunde Verzögerung dessen Quadratzahl ausgibt. Du kannst `.then` verwenden, um die Verzögerung zu realisieren.
2. Schreibe eine Funktion `squarePromise(n: number): Promise<number>`, die eine Zahl als Parameter entgegennimmt, die Zahl sofort ausgibt und mit 1 Sekunde Verzögerung dessen Quadratzahl *zurückgibt*. Schreibe diese Funktion in den folgenden zwei Varianten:
   1. als `async function`, die `await` auf einen Aufruf an `delay` anwendet. Tipp: Der Return-Wert von `async`-Funktionen ist immer ein `Promise` über dem Return-Wert der Funktion.
   2. als reguläre Funktion, die `.then` auf `delay` aufruft. Tipp: Das Ergebnis von `.then` ist immer ein neuer `Promise` über dem Return-Wert der Funktion in `.then`.
3. Schreibe eine Funktion `delayTimes(n: number, time: number): Promise<number>`, die mehrere `delay`-Aufrufe hintereinander ausführt, bevor sie das Ergebnis zurück gibt.
4. Erstelle die folgenden Funktionen, die mit mehreren Zahlen arbeiten:
   1. Erstelle eine Funktion `squareMultiple(nums: number[]): Promise<number>[]`, die eine Liste von Zahlen als Parameter entgegennimmt und für jede Zahl die Funktion `squarePromise` aufruft. Die Funktion gibt eine Liste von Promises zurück, die die Ergebnisse der Berechnungen enthalten. Rufe die Funktion mit der globalen Funktion `Promise.all` auf, um eine Liste von

[1] Es reichen eine kleine Hand voll Angaben und die Seite muss nicht schön gemacht werden - das kostet sonst nur wertvolle Zeit!

Promises `Promise<number>[]` in ein Promise von einer Liste `Promise<number[]>` zu verwandeln. Du kannst dann die Ergebnisse mit `.then` oder `await` abfragen.

2. Erstelle eine `async` Funktion `squareMultipleAsyncAwait(nums: number[]): Promise<number[]>`, die eine Liste von Zahlen als Parameter entgegennimmt in einer `for`-Schleife für jede Zahl die Funktion `squarePromise` mit `await` aufruft und in eine neue Liste speichert. Warum ist die vorherige Lösung mit `Promise.all` besser als diese Lösung?

Aufgabe 4 - TBA