Web-Technologien - Zwischenklausurthemen

Diese Fragen-/Themensammlung erhebt keinen Anspruch auf Vollständigkeit, sondern dient nur zu Demonstrationszwecken.

1. Basics: Protokolle & Tech-stack

a. Erklären Sie die zugrundeliegenden Mechanismen und Protokolle die benötigt werden, um einen GET-Request auf einen Webserver zu tätigen.

b. Was bedeutet `keep-alive` im Kontext von HTTP und ab welcher Version ist dieses Feature verfügbar?

c. Welche Informationen können über Header in HTTP übermittelt werden? Nenn Sie zwei konkrete Beispiele!

d. Welchen Zweck haben Statuscodes in HTTP? Geben Sie zwei Statuscodes an und erklären Sie deren Bedeutung!

2. HTML & CSS

a. Beantworten Sie folgende Fragen mit wahr / falsch:

```
i. HTML ist eine Touring-vollständige Programmiersprache.

ii. HTML-Dateien werden mithilfe einer speziellen Binärkodierung
geschrieben.

iii. HTML hat mit HTML5 einige Modernisierungen erhalten. Dies entspricht
auch der aktuell verwendeten Version von HTML.

iv. Tags können selbstschließend sein. Das bedeutet, dass es kein
schließendes Tag zu dem jeweils "öffnenden" Tag gibt.
```

b. Schreiben Sie das Grundgerüst einer HTML-Seite mit folgendem Inhalt:

- Überschrift ersten Grades mit dem Text "HTML is fun!"
- Absatz mit frei wählbarem Text
- Link mit dem Titel "Suchmaschine" der den Link https://duckduckgo.com/ in einem neuen Tab öffnet.

c. Formulieren Sie ein Tag, mit dessen Hilfe Sie die Datei `sunset.png` in der Datei `view.html` gemäß unten gegebener Dateistruktur einbinden können.

```
index.html
view.html
css
    |-> main.css
```

```
    |-> mobile.css
js
    |-> calculate.css
img
    |-> sunset.png
```

d. Geben Sie ein Tag an, um die Datei "mobile.css" gemäß der Dateistruktur in c einzubinden.

e. Erklären Sie das CSS Box-Modell sowie den Unterschied zwischen Inline- und Blockelementen.

f. Gegeben sei folgender body einer HTML-Seite:

```
<body>
    <header>
        <h1>My Personal Blog</h1>
        <p>... by Jon Doe</p>
        <div id="navbar">
            <a class="nav-item" href="index.html">
                Home
            </a>
            <a class="nav-item" href="about.html">
                About
            </a>
            <a class="nav-item" href="portfolio.html">
                Portfolio
            </a>
        </div>
    </header>

    <!-- main content -->
    <div>
        <p>
            <h3>First post</h3>
            <p>Some content here</p>
            <!-- post tags -->
            <span>interesting</span>
            <span>new</span>
            <span>life-advice</span>
        </p>
        <p>
            <h3>Second post</h3>
            <p>Some content here</p>
            <!-- post tags -->
            <span>interesting</span>
            <span>new</span>
            <span>life-advice</span>
        </p>
    </div>
</body>
```

Setzen Sie folgende Stylings mithilfe von CSS um:

- Der `header`-Bereich sollte weiße Schrift auf schwarzem Hintergrund beinhalten. Zudem ist zum nächsten Element ein Außenabstand von 4vh einzuhalten. Der Text ist zudem zentriert.
- Die `navbar` erhält die Hintergrundfarbe weiß mit einer Transparenz von 50%. Die Ecken sind abgerundet.
- Alle `nav-items` werden gleichmäßig über die Länge des Elternelements verteilt, sind nicht unterstrichen und ändern Ihre Farbe von schwarz auf grün wenn der Mauscursor unmittelbar darüber steht.
- Alle blogposts (`p` im zweiten `div`) sollten abwechselnd grau und weiß als Hintergrundfarbe besitzen und 66% der Bildschirmbreite einnehmen, sowie ober- und unterhalb einen Abstand von 3vh einhalten.

3. Basic JavaScript

a. Erklären Sie den Unterschied zwischen `var` und `let` in JavaScript.

Listing 1:

Gegeben sei folgendes Listing, welches die Basis für eine einfache ToDo-App darstellt:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ToDo</title>
</head>
<body>
    <h1>ToDolist-app</h1>
    <input type="text" placeholder="Enter todo item here .." id="new-todo-
txt" label="new-todo-txt">
    <input type="button" value="Add to list" id="add-item-btn">
    <input type="button" value="Clear list" id="clear-list-btn">

    <div id="todo-list">
        <div class="item">
            <p>Text of the item goes here</p>
            <input type="button" value="Delete item" class="delete-item-
btn">
        </div>
    </div>
</body>
</html>
```

b. Geben Sie ein Tag an, mit dessen Hilfe sie eine JavaScript-Datei einbinden können, die unter dem Pfad `js/custom/cms.js` zu finden ist. Markieren Sie außerdem die Stelle in Listing 1, in der Sie das Tag platzieren würden.

c. Implementieren Sie die Funktionalität des Buttons mit der `id add-item-btn`. Ein Klick auf den Button sollte den Text aus dem Element mit der `id new-todo-txt` nehmen und anschließend als neues Element in das `div` mit der `id todo-list` hinzufügen. Im `div` befindet sich bereits ein Element als Beispiel. Der Inhalt von `new-todo-txt` sollte dadurch gelöscht werden.

d. Stellen Sie sicher dass ein Klick auf `delete-item-btn` das jeweilige ToDo-item aus der Liste entfernt und `clear-list-btn` nach einem Klick die gesamte Liste leert.

e. Gegeben sei folgendes **Listing 2**:

```
    <table class="table table-striped">
        <thead class="thead-dark">
            <tr>
                <th scope="col">#</th>
                <th scope="col">Currency</th>
                <th scope="col">Type</th>
                <th scope="col">Amount in €</th>
                <th scope="col">Exchange rate</th>
                <th scope="col">Actions</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>#</td>
                <td>Bitcoin</td>
                <td>Buy</td>
                <td>100</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
            <tr>
                <td>#</td>
                <td>Bitcoin</td>
                <td>Buy</td>
                <td>500</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
            <tr>
                <td>#</td>
                <td>Shiba Inu</td>
                <td>Buy</td>
                <td>250</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
            <tr>
                <td>#</td>
                <td>Shiba Inu</td>
```

```
                <td>Buy</td>
                <td>200</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
            <tr>
                <td>#</td>
                <td>Shiba Inu</td>
                <td>Sell</td>
                <td>7000</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
            <tr>
                <td>#</td>
                <td>Shiba Inu</td>
                <td>Buy</td>
                <td>1000</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
            <tr>
                <td>#</td>
                <td>Bitcoin</td>
                <td>Sell</td>
                <td>50</td>
                <td>12345</td>
                <td><input type="button" class="btn btn-danger btn-sm
delete-btn" value="Delete"></td>
            </tr>
        </tbody>
    </table>
```

f. Benutzen Sie JavaScript um die Gesamtsumme der Tabelle aus Listing 2 zu berechnen. Beachten Sie dabei, dass Zeilen vom Typ `Buy` das Gesamtkapital verringern und deshalb abgezogen werden. Zeilen vom Typ `Sell` bringen Einnahmen und werden deshalb positiv hinzugezählt.

Beispiel:

```
Buy 100
Buy 200
Sell 800
Buy 1000
--> ergibt -500 (-100 + -200 + 800 + -1000)
```

g. Welchen Output erzeugt folgender JS Code:

```
function foo() {
    myVar = 100;
}

function bar() {
    let anotherOne = 1000;
}

console.log(myVar);
console.log(anotherOne);
```

h. Welchen Output erzeugt folgender JS Code:

```
if(0 == '' && null == undefined && !(NaN == NaN)) {
    console.log('weird stuff...');
} else {
    console.log('all good');
}
```

i. Schreiben Sie eine *higher-order function* `logAroundFunction`, welche eine auszuführende Funktion als Parameter bekommt und davor / danach ein `log` statement ausgibt.

j. Welchen Output erzeugt folgender JS Code:

```
function cMult(x, y) {
    const multiplyByAdding = function (num, times) {
        result = 0;

        for(let i = 1; i <= times; i++) {
            result = result + num;
        }
    }

    if(y === undefined) {
        return (ry) => {
            multiplyByAdding(x, ry);
            return result;
        }
    } else {
        multiplyByAdding(x, y);
        return result;
    }
}

let multResult = cMult(10);
console.log(multResult);
console.log(multResult(10));
```

k. Schreiben Sie eine Konstruktorfunktion, welche ein Objekt `Person` mit folgenden Attributen erzeugt:

- firstname
- lastname
- gender
- age
- isStudent
- isWorking Außerdem sollte für jede Person eine Methode `introduce()` vorhanden sein. Diese gibt `Hello, my name is <name>` auf der Konsole aus. Achten Sie darauf, dass die Methode nicht für jedes Objekt extra im Speicher abgelegt werden muss.

l. Setzen Sie die Konstruktorfunktion aus k mithilfe von ES6 class-Syntax um.

m. Welchen Output erzeugt folgender JS code, wenn die Person mit Namen 'Susi' ausgewählt ist und der Benutzer auf den Button klickt?

```
-- index.html
<input type="button" value="introduce" id="introduce-btn" name="intr-btn">
<select id="persons" name="prsn">
</select>

-- foo.js (eingebunden in index.html)
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    introduce() {
        console.log(`Hello my name is ${this.name} and I am ${this.age}
years old`);
    }
}

function parsePersonArrayToOptionArray(personObjects) {
    let el;
    return personObjects.map((person, personIdx) => {
        el = document.createElement('option');
        el.setAttribute('value', personIdx);
        el.innerText = person.name;
        return el;
    });
}

let personObjects = [];
personObjects.push(new Person('Hugo', 22));
personObjects.push(new Person('Susi', 25));

let personDropdown = document.querySelector('#persons');
parsePersonArrayToOptionArray(personObjects).forEach(personElement => {
    personDropdown.appendChild(personElement);
});
```

```
document.querySelector('input[name="intr-btn"]').addEventListener('click',
personObjects[personDropdown.value].introduce);
```

n. Wie würde sich die Ausgabe verändern wenn der EventListener den Aufruf von `introduce` in einer Arrow-function kapselt?

4. Advanced JavaScript

a. Welche Konzepte kennen Sie um mit Asynchronität in JavaScript umzugehen?

b. Erklären Sie wie Asynchronität in JavaScript umgesetzt ist!

c. Schreiben Sie eine Funktion `executeFunctionNTimesAfterDelay` welche drei Parameter entgegennimmt:

- eine Funktion `func`
- eine Zahl `numberOfExecutions`
- eine Zahl `delayInMs` `func` sollte dabei `numberOfExecutions` mal ausgeführt werden. Vor der ersten Ausführung wird dabei `delayInMs` Millisekunden gewartet. Die Funktion sollte einen Fehler werfen wenn `numberOfExecutions > 100` ist. In diesem Fall soll `Execution threshold overflow` ausgegeben werden. Im Erfolgsfall soll nach der vollständigen Ausführung `Function executed <n> times` ausgegeben werden.

d. Schreiben Sie JS code der einen `POST`-request an `http://www.loginservice.xyz/login` sendet. Der request-body sollte folgendes beihalten:

```
    {
        username: 'user',
        password: '123456',
        maxLoginAttempts: 5
    }
```

5. node.js

a. Beantworten Sie folgende Fragen mit `wahr` oder `falsch`:

```
    i. Node.js transpiliert JS code während der Auführung in C-code, damit
dieser im Browser ausgeführt werden kann.
    ii. Node.js wird häufig für Backends von Applikationen eingesetzt.
    iii. Google Chrome und node.js verwenden dieselbe JavaScript-engine.
    iv. Das Tool `npm` ist für die Verwaltung externer Pakete in node-
Applikationen zuständig.
    v. Node.js kann auch ohne `npm` verwendet werden.
```

b. Schreiben Sie einen Webservice für die Verwaltung von ToDo-Listen, welcher folgende Funktionalität beinhält:

- Erreichbar über Port 8080
- `/` gibt `ToDoList service is up and running` zurück
- Anlegen, Namensänderung und löschen von ToDo-Listen
- Anlegen und löschen von Elementen in einer ToDo-Liste
- Abfrage aller Elemente in einer ToDo-Liste
- Abfrage aller Elemente aller ToDo-Listen
- Achten Sie darauf, die REST-Prinzipien nicht zu verletzen!