Klausur „Webapplikationen“, Angewandte Informatik, HS Fulda	
Name:	Matrikelnummer:

Webapplikationen
21.07.2021

Name: _____ **LÖSUNG** _____

Vorname: _____

Matrikelnr.: _____

Semester: _____

Aufgabe	1	2	3	4
Punkte	*15*	*15*	*15*	*15*

Summe (max. 60 Punkte): _____

Note: _____

Erlaubtes Hilfsmittel: ein einseitig und handschriftlich beschriebenes DIN-A4-Blatt

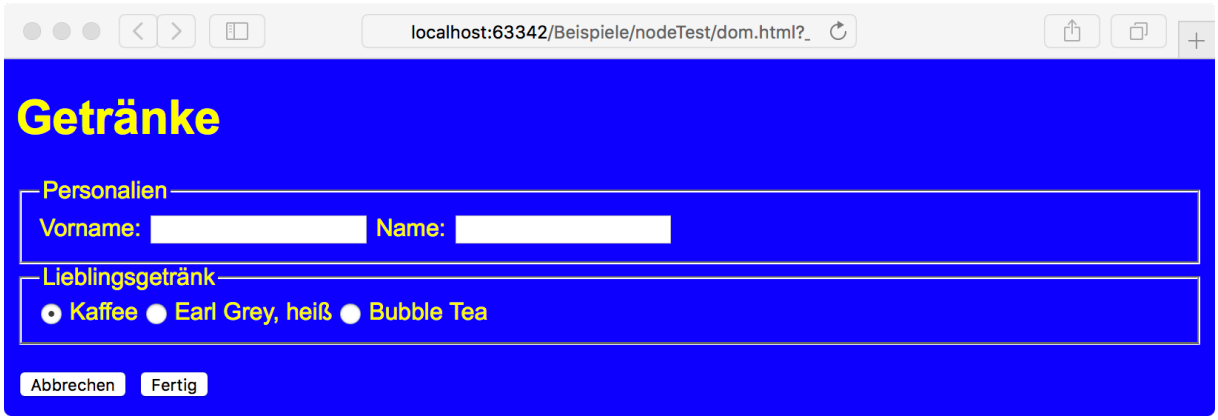
Viel Erfolg!

Klausur „Webapplikationen“, Angewandte Informatik, HS Fulda

Name: Matrikelnummer:

1. Aufgabe (HTML und DOM) 15 Punkte

Zeichnen Sie für den nachfolgenden HTML-Code möglichst genau ein, wie die entsprechende Webseite aussehen wird. Skizzieren Sie (auf der Folgeseite) den dazugehörigen DOM-Tree!



4

```
<html>
<head>
<title>Klausur Web-Apps SS2021</title>
<style>
  body {
    background-color: #0000ff;
    color: #ffff00;
    font-family: Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Getr&auml;nke</h1>
<form action="getraenke.php" method="get">
<fieldset>
<legend>Personalien</legend>
Vorname: <input type="text" name="vorname">
Name: <input type="text" name="nachname">
</fieldset>
<fieldset>
<legend>Lieblingsgetr&auml;nk</legend>
<input type="radio" name="getraenk" value="k" checked> Kaffee
<input type="radio" name="getraenk" value="t"> Earl Grey, hei&szlig;
<input type="radio" name="getraenk" value="b"> Bubble Tea
</fieldset>
<p>
<input type="reset" value="Abbrechen">
<input type="submit" value="Fertig">
</p>
</form>
</body>
</html>
```

Klausur „Webapplikationen“, Angewandte Informatik, HS Fulda	
Name:	Matrikelnummer:

Wie lautet der Query-String, wenn Sie das Formular mit Ihrem Namen und Getränkewunsch ausfüllen und absenden?

[?]vorname=Yvonne&nachname=Jung&getraenk=k
// bzw. natürlich eigener Name u. Getränkewunsch

1

Stylen Sie nun den Absatz, der die beiden Buttons vom Typ Reset und Submit beinhaltet, so dass er von einer dicken roten Umrandung umgeben ist, deren Ecken leicht abgerundet sind. Die Buttons selbst sollen mittig darin platziert werden, wobei insbes. nach oben und unten 10 Pixel Platz zum Rand gelassen werden sollen. Der Absatz selbst soll nur die Hälfte der Gesamtbreite ausnutzen, nach oben und unten einen Abstand von 5 Pixeln haben und dabei auch mittig auf der Seite positioniert werden.

```
p {
  border: 2px solid red;
  border-radius: 5px;
  text-align: center;
  padding: 10px;
  width: 50%;
  margin: 5px auto;
}
```

4

DOM-Tree:

```
html
  head
    title
      #text
    style
      #text
  body
    h1
      #text
    form
      fieldset
        legend
          #text
        #text
        input
        #text
        input
      fieldset
        legend
          #text
        input
        #text
        input
        #text
        input
        #text
      p
        input
        input
```

// Einrückungen symbolisieren Verbindungen

6

2. Aufgabe (JavaScript-Programmierung) 15 Punkte

Schreiben Sie in JavaScript eine Klasse **Vektor3D** für dreidimensionale Vektoren im $\mathbb{R}^3$ mit einem geeigneten Konstruktor, der die Werte für *x*, *y*, *z* entgegennimmt. Die Koordinaten eines Vektors sollen dabei intern in einem eindimensionalen Array abgespeichert werden!

Die Klasse soll zudem die beiden Methoden `anzeigen()` und `berechneKreuzprodukt(v)` haben. Erstere soll lediglich den aktuell gespeicherten 3D-Vektor auf der Konsole ausgeben. Die Berechnung des Kreuzproduktes zweier Vektoren ist bekanntlich nur im $\mathbb{R}^3$ definiert, weswegen hier auch eventuelle Fehlerfälle abzufangen sind, wobei die folgende Rechenregel umzusetzen ist: $\vec{c} = \vec{a} \times \vec{b} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_2 - b_1 a_2 \\ b_0 a_2 - a_0 b_2 \\ a_0 b_1 - b_0 a_1 \end{pmatrix}$. Der Rückgabewert ist ein neues Objekt vom Typ Vektor3D, dessen Datenfeld das Ergebnis der Berechnung hält.

Instanziieren Sie anschließend die zwei Vektoren $\vec{a} = (1,0,0)^T$ und $\vec{b} = (0,1,0)^T$, berechnen das Kreuzprodukt mit Hilfe der o.g. Methode (nur den entsprechenden Code hinschreiben, nicht selbst rechnen), was einem dritten Vektor $\vec{c}$ zugewiesen wird, der zuletzt anzuzeigen ist.

```
class Vektor3D {
    constructor(x=0, y=0, z=0) {
        this._val = [x, y, z];
    }

    berechneKreuzprodukt(v) {
        if (!v || !v._val || v._val.length < 3)
            return null;
        let w = new Vektor3D();
        w._val[0] = this._val[1] * v._val[2] - v._val[1] * this._val[2];
        w._val[1] = this._val[2] * v._val[0] - v._val[2] * this._val[0];
        w._val[2] = this._val[0] * v._val[1] - v._val[0] * this._val[1];
        return w;
    }
    anzeigen() {
        console.log(this._val.join(' '));
    }
}

/*
var Vektor3D = function(x, y, z) {
    this._val = [x || 0, y || 0, z || 0];
};

Vektor3D.prototype.berechneKreuzprodukt = function(v) {
    var w = new Vektor3D();
    w._val[0] = this._val[1] * v._val[2] - v._val[1] * this._val[2];
    w._val[1] = this._val[2] * v._val[0] - v._val[2] * this._val[0];
    w._val[2] = this._val[0] * v._val[1] - v._val[0] * this._val[1];
    return w;
};

Vektor3D.prototype.anzeigen = function() {
    console.log(this._val.join(' '));
};
*/

const a = new Vektor3D(1, 0 , 0), b = new Vektor3D(0, 1, 0);
const c = a.berechneKreuzprodukt(b);
c.anzeigen();
```

3. Aufgabe (JSON und DOM) 15 Punkte

Nehmen Sie an, Sie hätten eine JSON-Datei mit folgendem Inhalt vom Server erhalten, um dynamisch zur Laufzeit neue HTML-Fragmente Ihrem HTML-Dokument hinzuzufügen.
Nehmen Sie weiter an, dass Sie dazu das JSON in ein JavaScript-Objekt umgewandelt und an eine Variable namens `obj` zugewiesen haben. Wie würde der JavaScript-Code aussehen, wenn Sie stattdessen das JS-Objekt direkt im Code angelegt hätten? Sie dürfen dafür das untenstehende JSON entsprechend verändern bzw. korrigieren.

```
let obj =
{
  "section": {
    "h2": "Haustiere",
    "p": "Viele Menschen haben ein Haustier. Beliebt sind:",
    "ul": [
      {"li": "Hund"},
      {"li": "Katze"},
      {"li": "Kaninchen"},
      {"li": "Meerschweinchen"}
    ]
  }
}
;
// Ganz wichtig: alle Schlüssel (z.B. h2 oder p) je OHNE Anführungszeichen!
```

3

Wie Sie sich evtl. schon gedacht haben, sollen die Schlüssel des resultierenden JS-Objekts den gleichnamigen HTML-Elementen entsprechen (z.B. <p>), wobei Werte mit reinem Text einfachen Textknoten entsprechen.
Achtung: mit dieser vereinfachten Syntax lassen sich nach Konvertierung in JavaScript leider noch keine beliebigen HTML-Dokumente generieren. Erklären Sie, wo hier das Problem liegt!

Bei beliebigen HTML-Dokumenten kann nach einem Element bestimmten Typs (z.B. <p>) später nochmal ein solches Element kommen. In JSON bzw. bei JavaScript-Objekten müssen die Schlüssel aber eindeutig sein; d.h., käme hier nochmal auf gleicher Ebene ein weiteres „p“, dann würde dessen Wert den des ersten „p“'s überschreiben, was nicht gewollt ist.

2

Wie erwähnt, soll aus obigem JSON dynamisch der korrespondierende HTML-Code erzeugt werden. Vervollständigen Sie dazu die auf der nächsten Seite begonnene rekursive Funktion, die mit Hilfe einer For-in-Schleife über das oben erwähnte JS-Objekt `obj` iteriert, um die entsprechenden HTML-Elemente zu erzeugen.
Möchte man damit z.B. alles an das Ende der HTML-Datei anfügen, so sähe der initiale Aufruf wie folgt aus: `addFragmentToDOM(obj, document.body);`

Der Einfachheit halber wird hier jedes neu erzeugte HTML-Element direkt in den DOM-Tree der Webseite eingefügt. Begründen Sie zunächst kurz, ob dies eine gute Idee ist!

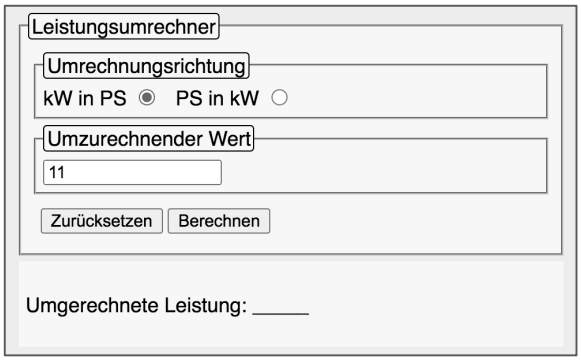
Nein, da dies nicht performant ist. Besser ist es, erst den kompletten neuen Teilbaum zu erzeugen und diesen dann auf einmal in den DOM-Tree zu hängen, weil dann z.B. das ganze Layout usw. nur einmalig neu berechnet werden muss.

2

```
function addFragmentToDOM(obj, parent) {
    for (const key in obj) {
        let elem = document.createElement(key);
        parent.appendChild(elem);
        if (typeof obj[key] === "object") {
            if (obj[key] instanceof Array) {
                for (let i=0; i<obj[key].length; i++) {
                    addFragmentToDOM(obj[key][i], elem);
                }
            }
            else {
                addFragmentToDOM(obj[key], elem);
            }
        }
        else {
            elem.innerText = obj[key];
        }
    }
}
```

8

4. Aufgabe (Ajax und Node.js) 15 Punkte



Implementieren Sie mit Hilfe von JavaScript und Ajax zunächst den clientseitigen Teil einer kleinen Webanwendung zum Umrechnen von Kilowatt in Pferdestärke und umgekehrt (vgl. Abb.).

Dabei wird bei Klick auf den Button „Berechnen“ über Ajax per GET ein *Node.js*-Script aufgerufen (das zum Testen auf Ihrem lokalen Server läuft; s. nächste Seite), das zum einen den umzurechnenden Wert in einem Inputelement namens „leistung“ erwartet (in der Abb. exemplarisch ‚11’ eingetragen). Zum anderen wird die Umrechnungsrichtung (kW in PS oder umgekehrt) in einem booleschen Parameter „kWinPS“ (wofür die Eigenschaft „checked“ von Radiobuttons nützlich ist) erwartet. Der umgerechnete Wert wird dann direkt auf der HTML-Seite (statt der Unterstriche) angezeigt, ohne die Seite dabei wieder vollständig neu zu laden.

Setzen Sie hier zunächst den *clientseitigen* Teil um, wobei Sie lediglich die für die Umsetzung notwendigen HTML-Elemente aus dem <body> (kein CSS!) sowie den JavaScript-Code mit dem Ajax-Request (XMLHttpRequest oder *fetch()*) aufzuschreiben brauchen.

```
<form>
    kW in PS <input type="radio" name="wat" id="kw" checked>
    PS in kW <input type="radio" name="wat" id="ps">
    <input type="text" id="val" name="leistung" value="11">
    <input type="reset" value="Zurücksetzen">
    <input type="button" id="btn" value="Berechnen">
</form>
<div>Umgerechnete Leistung: <span id="erg">_____</span></div>
```

4

```
//document.querySelector('#btn').addEventListener("click", function() {
document.getElementById('btn').addEventListener("click", /*async*/ function() {
    let kWinPS = document.getElementsByName('wat')[0].checked;
    let leistung = document.getElementById('val').value;
    let query = "http://localhost:35668/?leistung=" +
                leistung + "&kWinPS=" + kWinPS;

    /* let xhr = new XMLHttpRequest();
    xhr.onload = function() {
    document.getElementById("erg").innerHTML =
        (+this.responseText).toFixed(0);
    };
    xhr.open("GET", query, true);
    xhr.send(); */

    fetch(query).then(response => response.text())
                .then(data => document.getElementById("erg").innerHTML =
                    (+data).toFixed(0));

    /* const res = await fetch(query);
    const str = await res.text();
    document.getElementById("erg").innerHTML = (+str).toFixed(0); */
});
```

6

Der *serverseitige* Teil wurde im Wesentlichen schon fertiggestellt, ist allerdings noch fehlerhaft. Korrigieren Sie daher bitte noch den nachfolgenden JavaScript-Code.[1] Beachten Sie für die Umrechnung die folgende Beziehung: 1 PS ≈ 0,7355 kW (bzw. 1 kW ≈ 1,3596 PS). Überprüfen Sie ggfs. auch noch einmal, ob Ihr clientseitiger Teil der Webanwendung auch dazu passt.

```
const http = require('http');

const server = http.createServer();
const port = 35668;

server.on('request', function(req, res) {

    let query = "";
    let ind = req.url.indexOf('?');

    if (ind >= 0 && ++ind < req.url.length) {
        query = req.url.substr(ind);
    }
    else {
        console.error("Fehler in URL: " + req.url);
        return;
    }

    const params = new URLSearchParams(query);

    if (params.has("leistung") &&
        params.has("kWinPS") ) {

        res.writeHead(200, {
            'Content-Type': 'text/plain',
            'Access-Control-Allow-Origin': '*'
        });

        let lwert = +params.get("leistung");
        let kWinPS = params.get("kWinPS") === "true";

        if (kWinPS) {
            lwert *= 1.3596;
        }
        else {
            lwert *= 0.7355;
        }

        res.write(lwert.toString());
        res.end();
    }
});

server.listen(port);
```

5

[1] Anmerkung: *URLSearchParams* aus der neuen URL API ersetzt das veraltete URL-Modul von Node und wird mit dem Query-String, der wie gehabt in `req.url` zu finden ist, initialisiert (die entsprechende Code-Zeile ist korrekt). Mit der Methode *has()* kann man nun überprüfen, ob ein gegebener Parameter existiert und mit *get()* kann man den (als String abgespeicherten) Wert eines Parameters auslesen.