# Web Technologies: Midterm exam topics

Note: This collection of topics/questions doesn't cover the entire pool of topics, but rather serves for demonstration purpose.

## 1. Basics: Protocols & Tech-stack

a. Explain underlying mechanisms and protocols that are needed to issue an HTTP GET request to a web server.

b. What does `keep-alive` mean in the context of HTTP and in which version was it first usable?

c. What is the purpose of HTTP-headers and which information can be conveyed by them? List two concrete examples.

d. Explain the concept of status codes in HTTP. List two status codes and explain their meaning.

## 2. HTML & CSS

a. Answer the following questions with `true` / `false`.

```
i. HTML is a touring complete programming language.
ii. HTML files need to be created in a hex-editor.
iii. HTML received some modernisation with the release of HTML5. This is
the currently used version of HTML.
iv. Tags in HTML can be self-closing. That means that there is no
dedicated closing tag to an opening tag.
```

b. Create a bare-bones HTML page with the following content:

- First-order heading with the text "HTML is fun".
- Paragraph with arbitrary text
- Link with the title "Search engine" that opens the URL "https://duckduckgo.com/" in a new tab.

c. Formulate a tag to embed the file `sunset.png` in the file `view.html` according to the given file structure below.

```
index.html
view.html
css
    |-> main.css
    |-> mobile.css
js
    |-> calculate.css
img
    |-> sunset.png
```

d. Write down a tag to include the file `mobile.css` according to the file structure in question c.

e. Explain the CSS box model and outline differences between inline- and block elements.

f. Given the following `body` on a page:

```html
<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 class="blogpost">
            <h3>First post</h3>
            <p>Some content here</p>
            <!-- post tags -->
            <span>interesting</span>
            <span>new</span>
            <span>life-advice</span>
        </p>
        <p class="blogpost">
            <h3>Second post</h3>
            <p>Some content here</p>
            <!-- post tags -->
            <span>interesting</span>
            <span>new</span>
            <span>life-advice</span>
        </p>
    </div>
</body>
```

Apply CSS to achieve the following styles:

- The `header` section should contain white colored font on a black background. The margin to the next element should be 4vh. Text should be centered.
- The `navbar` has white background color with a transparency of 50%. The corners are rounded with a radius of 10px.

- All `nav-items` are distributed evenly across the width of the parent element, are not underlined and turn their font color from black to green if hovered.
- All blogposts (`p` elements in second `div`) should have an alternating background-color of grey and white. Further, they should occupy 66% of the screen width and maintain a top and bottom margin of 3vh.

## 3. Basic JavaScript

a. Explain the difference between `var` and `let` in JavaScript

**Listing 1**

```
<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>
```

b. Write down a tag you would use to include a JavaScript file that exists in `js/custom/cms.js`.

c. Implement the functionality of the button with id `add-item-btn`. Clicking on the button should grab the text from element with id `new-todo-txt` and insert it into the `div` with id `todo-list`. Inside `todo-list`, there is already a sample item in listing 1 that shows you what the inserted item should look like. Also, the text in `new-todo-txt` should be cleared after the item has been added.

d. Ensure that a click on `delete-item-btn` deletes the respective ToDo-item and `clear-list-btn` clears the entire ToDo-list.

Given the following **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>Sell</td>
      <td>4000</td>
      <td>78910</td>
      <td>
        <input
          type="button"
          class="btn btn-danger btn-sm delete-btn"
          value="Delete"
        />
      </td>
    </tr>
  </tbody>
</table>
```

f. Use JavaScript to calculate the total balance of investments from the table in Listing 2. Note that rows with a `Buy` decrease the overall balance because money is spent, `Sell` increases the overall balance because money is gained.

Example:

```
Buy 100
Buy 200
Sell 800
--> total of 500 (-100 -200 +800)
```

g. What is the output of the following JS code:

```
function foo() {
    myVar = 100;
}
function bar() {
    let anotherOne = 1000;
}
console.log(myVar);
console.log(anotherOne);
```

h. What is the output of the following JS code:

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

i. Write a higher-order function with name `logAroundFunction` that receives an arbitrary `function` as a parameter and adds a `log` statement before / after executing the given function.

j. What is the output of the following 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. Write a constructor function that creates an Object `Person` with the following attributes:

- firstname
- lastname
- gender
- age

- isStudent
- isWorking

Besides, every person should have a method `introduce()` that prints `Hello, my name is <firstname> <lastname>.` to the console. Take care that the method isn't stored separately in memory for every created object.

l. Translate the constructor function from question k with ES6 class syntax.

m. What is the output of below JS code, if the `Person` with name `Susi` is selected and the user clicks on the button?

```
-- 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. What would the output be if the `EventListener` that calls `introduce` gets encapsulated into an arrow-function? Also explain why the output would change at all.

## 4. Advanced JavaScript

a. Explain which concepts exist in JavaScript to handle asynchronuos code.

b. Explain how asynchronous execution is implemented into the JavaScript engine.

c. Write a function `executeFunctionNTimesAfterDelay` that takes three parameters

- a function `func`
- a number `numberOfExecutions`
- a number `delayInMs`

The given `func` should get executed `numberOfExecutions` times. Your implementation should wait `delayInMs` milliseconds before the first execution of `func`. An error should be signaled if `numberOfExecutions > 100` by outputting `Execution threshold overflow` onto the console. In the case of successful execution, the text `Function executed <n> times` should get printed.

d. Write JS code that issues a `POST` request to `http://www.loginservice.xyz/login`. The request body should contain the following content:

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

## 5. node.js

a. Answer the following questions with `true` / `false`.

```
i. Node.js transpiles JS code during execution to C code, so that it can
get executed by the browser.
ii. Node.js is usually used for implementing application backend systems.
iii. Google Chrome and node.js use the same JavaScript engine.
iv. The tool `npm` is responsible for managing external packages and
dependencies in node applications.
v. Node.js can also be used without `npm.
```

b. Write a web service for management of ToDo lists that contains the following functionality:

- Accessible through port 8080
- `/` returns the text `ToDo list service is up and running`
- Create, change name and delete ToDo-lists

- Create, change & delete elements of ToDo-lists
- Retrieve all elements of a given ToDo list
- Retrieve all elements of all ToDo lists
- Pay attention to not hurt any of the REST principles when designing your API!