Week 4 Study Guide

Notes

DOM vs BOM

- the `Document Object Model` is the hierarchy/representation of the objects that comprise a document on the web (i.e. how all elements in a document are organized). The DOM is a part of the `Browser Object Model`, the hierarchy/representation of all browser objects associated with the web browser.

Browser Diagram

User Interface
v
Browser Engine ---> Data Persistence
v
Rendering Engine
v v v
Networking JS interpreter UI backend
==
- `User Interface` : everything in browser interface except requested page content
- `Browser Engine` : manages interactions btwen UI and rendering engine
- `Rendering Engine` : renders requested page content, parsing HTML & CSS
- `Networking` : handles network calls, i.e. HTTP requests
- `JS Interpreter` : parses & executes JS code
- `UI Backend` : used for drawing basic widgets, using operating system UI methods
- `Data Persistence` : persistence of data stored in browser, i.e. cookies

```
- A browser's main role in the request/response cycle is:
    1. Parsing HTML, CSS, JS
    2. Rendering that information to the user by constructing a DOM tree and rendering it.
```

Request-Response Diagram

```
Click --->          Request --->        Server <----> Data
    Browser
Page <---           <--- Response

Your Computer       The Internet        Data Center
```

Window API

- the `Window API` includes the methods and properties that you can use on the `window object` , the core of the `BOM` .
- using the `Window API` to resize the browser window:

  ```
  //opens a new window
  newWindow = window.open("url", "name", "width=100, height=100")
  //resizes new window
  newWindow.resizeTo(500,500)
  //also resizes by given amount; use `-` to shrink
  newWindow.resizeBy(xDelta, yDelta)
  ```

- the `context` of an `anonymous function` fun in the browser will be the `window object` . Remember that every function has a context, which we can think of as which object OWNS the function, and context is most often determined by how a function is invoked.

Running Scripts

- Insert a script via a .js document into an .html document:

  ```
  <html>
  <head>
      <script type="text/javascript" src="dom-ready-script.js"></script>
  </head>
  <body></body>
  <html></html>
  </html>
  ```

- Run the script on DOMContentLoaded (when the doc has been loaded, but without waiting for stylesheets, images, and subframes):

  ```
  window.addEventListener("DOMContentLoaded", event => {
      console.log("This script loaded when the DOM was ready.");
  });
  ```

- Run the script on page load using window.onload (wait for EVERYTHING to load):

  ```
  window.onload = () => {
      console.log(
          "This script loaded when all the resources and the DOM were ready."
      );
  };
  ```

- Three ways to prevent script from running until page loads:
  1. Use `DOMContentLoaded`
  2. Place `script tag` at very bottom of HTML file
  3. Add attribute like `async` or `defer`

async vs defer

- `<script>` without any attributes will pause HTML parsing, and a request will be made to fetch the file (if it

is external). The script will be executed before parsing is resumed.
- `async` downloads the file during HTML parsing and will pause the HTML parser to execute it once it has downloaded.
- `defer` downloads the file during HTML parsing, but will only execute it after the parser has completed.
- the standard is to use `async`, then `defer`.

Cookies vs Web Storage API

- Cookies - stores stateful info about a user, transfers data to server, under 4KB storage limit
- `sessionStorage` - stores data only for a session, until browser window/tab is closed, does not transfer data to server, 5MB storage limit.
- `localStorage` - stores data w/no expiration date, does not transfer data to server, deleted when browser cache is cleared; maximum storage limit
- Create a cookie:

  ```
  const firstCookie = "favoriteCat=million";
  document.cookie = firstCookie;
  const secondCookie = "favoriteDog=bambi";
  document.cookie = secondCookie;
  document.cookie; // Returns "favoriteCat=million; favoriteDog=bambi"
  ```

- Delete a cookie by setting a cookie's expiration date to the past (or delete in Developer Tools):

  ```
  document.cookie = "favoriteCat=; expires = Thu, 01 Jan 1970 00:00:00 GMT";
    document.cookie; // Returns "favoriteDog=bambi"
  ```

- Create localStorage data:

  ```
  //set new localStorage item
  localStorage.setItem("firstThing", "firstValue");
  //retrieve that localStorage item
  localSTorage.getItem("firstThing");
  ```

- When to use the Web Storage API?
  - shopping cart
  - input data on forms
  - info on user i.e. preferences or buying habits
- When to use cookies:
  - Session cookie, stores session info on user login/validation (lost once browser is closed unless you use a persistent cookie)
- You can view cookies and web storage info with Developer Tools (inspect -> Application tab).

Learning Objectives

BROWSER BASICS & STORAGE

1. ✓ Explain the difference between BOM (browser object model) and the DOM (document object model).
2. ✓ Given a diagram of all the different parts of the Browser, identify each part.
3. ✓ Use the Window API to change the innerHeight of a user's window.
4. ✓ Identify the context of an anonymous function running in the Browser (the window).
5. ✓ Given a JS file and an HTML file, use a script tag to import the JS file and execute the code therin when all elements on a page load (using DOMContentLoaded).
6. ✓ Given a JS file and an HTML file, use a script tag to import the JS file and execute the code therein when the page loads.
7. ✓ Identify three ways to prevent JS code from executing until an entire HTML page is loaded.
8. ✓ Label a diagram on the Request/Response cycle.
9. ✓ Explain the Browser's main role in the request/response cycle (1. Parsing HTML, CSS, JS; 2. Rendering that information to the user by constructing a DOM tree and rendering it.)
10. ✓ Given several detractors - identify which real-world situations could be implemented with the Web Storage API (shopping cart, forms savings inputs, etc.).
11. ✓ Given a website to visit that depends on cookies (like Amazon), students should be able to go to that site, add something to their cart, and then delete that cooking using Chrome Developer tools in order to empty their cart.
12. ✓ Write JS to store the value "I ♡ falafel" with the eky "eatz" in the browser's local storage.
13. ✓ Write JS to read the value stored in local storage for the key "paper-trail".

ELEMENT SELECTION

1. Given HTML that includes `<div id="catch-me-if-you-can>HI!</div>` , write a JS statement that stores a reference to the HTMLDivElement with the id "catch-me-if-you-can" in a variable named "divOfInterest".

   ```
   let divOfInterest = document.getElementById("catch-me-if-you-can");
   ```

2. Given HTML that includes seven SPAN elements each with the class "cloudy", write a JS statement that stores a reference to a NodeList filled with references to the seven HTMLSpanElements in a variable named "cloud

   ```
   let cloudyNodes = document.querySelectorAll("span.cloudy");
   ```

3. Given an HTML file with HTML, HEAD, TITLE, and BODY elements, create and reference a JS file that in which the JS will create and attach to the BODY element an H1 element with the id "sleeping-giant" with the content "Jell-O, Burled!".

   ```
   <script type="text/javascript" src="location.file"></script>
   ```

```
let newHeader = document.createElement("h1");
newHeader.setAttribute("id", "sleeping-giant");
newHeader.innerHTML = "Jell-O, Burled!";
//const newContent = document.createTextNode("Jell-O, Burled!");
document.body.appendChild(newHeader);
```

4. Given an HTML file with HTML, HEAD, TITLE, SCRIPT, and BODY elements with the SCRIPT's SRC attribute referencing an empty JS file, write a script in the JS file to create a DIV element with the id "lickable-frog" and add it as the last child to the BODY element.

```
<script type="text/javascript" src="location.file"></script>


let newDiv = document.createElement("div");
newDiv.setAttribute("id", "lickable-frog");
document.body.appendChild(newDiv);
```

5. Given an HTML file with HTML, HEAD, TITLE, SCRIPT, and BODY elements with no SRC attribute on the SCRIPT element, write a script in the SCRIPT block to create a UL element with no id, create an LI element with the id "dreamy-eyes, add the LI as a child to the UL element, and add the UL element as the first child of the BODY element.

```
<script type="text/javascript">
    let newList = document.createElement("ul");
    let newItem = document.createElement("li");
    newItem.setAttribute("id", "dreamy-eyes");
    newList.appendChild(newItem);
    document.body.prepend(newList);
</script>
```

6. Write JS to add the CSS class "i-got-loaded" to the BODY element when the window fires the DOMContentLoaded event.

```
document.addEventListener("DOMContentLoaded", event => {
   document.body.className("i-got-loaded");
});
```

7. Given an HTML file with a UL element with the id "your-best-friend" that has six non-empty ILs as its children, write JS to write the content of each LI to the console.

```
let parentList = document.getElementById("your-best-friend");
let childNodes = parent.childNodes;
for (let value of childNodes.values()) {
    console.log(value);
}
```

8. Given an HTML file with a UL element with the id "your-worst-enemy" that has no children, write JS to construct a string that contains six LI tags each containing a random number and set the inner HTML

property of ul#your-worst-enemy to that string.

```
const getRandomInt = max => {
    return Math.floor(Math.random() * Math.floor(max));
}
const liArr = [];
for (let i = 0, i < 6, i++) {
    liArr.push("<li>" + getRandomInt(10) + "</li>")
}
const liString = liArr.join(" ");
const listElement = document.getElementById("your-worst-enemy")
listElement.innerHTML = liString;
```

9. Write JS to update the title of the document to the current time at a reasonable interval such that it looks like a real clock.

```
<title id="title"></title>


const title = document.getElementById("title");
const time = () => {
    const date = new Date();
    const seconds = date.getSeconds();
    const minutes = date.getMinutes();
    const hours = date.getHours();

    title.innerHTML = `${hours}:${minutes}:${seconds}`
};
setInterval(time, 1000);
```

EVENT HANDLING

1. Given an HTML page that includes `<button id="increment-count">I have been clicked <span id="clicked-count">0</span> times</button>` , write JS that increases the value of the content of `span#clicked-count` by 1 every time `button#increment-count` is clicked.

```
let incrementButton = document.getElementById("increment-count");
let incrementSpan = document.getElementById("clicked-count");
let count = 0
incrementButton.addEventListener("click", event => {
    count++
    incrementSpan.innerHTML = count
});
```

2. Given an HTML page that includes `<input type="checkbox" id="on-off"><div id="now-you-see-me">Now you see me</div>` , write JS that sets the display of div#now-you-see-me to "none" when input#on-off is checked and "block" when input#on-off is not checked.

```
let inputBox = document.getElementById("checkbox");
let divSee = document.getElementById("now-you-see-me");
inputBox.addEventListener("click", event => {
    if (inputBox.checked) {
        divSee.style.display = "block";
    } else {
        divSee.style.display = "none";
    };
});
```

3. Given an HTML file that includes `<input id="stopper" type="text" placeholder="Quick! Type STOP">` , write JS that will change the background color of the page to cyan five seconds after a page loads unless the field input#stopper contains only the text "STOP".

```
let inputStopper = document.getElementById("stopper");
const stopCyanMadness = () => {
    if (inputStopper.value !== "STOP") {
        document.body.style.backgroundColor = "cyan";
    }
};
setTimeout(stopCyanMadness, 5000);
```

4. Given an HTML page that includes `<input type=”text” id=”fancypants”>` , write JS that changes the background color of the textbox to #E8F5E9 when the caret is in the textbox and turns it back to its normal color when focus is elsewhere.

```
const input = document.getElementById("fancypands");
input.addEventListener("focus", event => {
    event.target.style.backgroundColor = "#E8F5E9";
});
input.addEventListener("blur", event => {
    event.target.style.backgroundColor = "initial";
})
```

5. Given an HTML page that includes a form with two password fields, write JS that subscribes to the forms submission event and cancels it if the values in the two password fields differ.

```
let form = document.getElementById("signup-form");
let passwordOne = document.getElementById("password);
let passwordTwo = document.getElementById("password);
form.addEventListener("submit", event => {
    if (passwordOne.value !== passwordTwo.value) {
        event.preventDefault();
        alert("Passwords must match!");
    } else {
        alert("The form was submitted!");
    }
})
```

6. Given an HTML page that includes a div styled as a square with a red background, write JS that allows a user to draf the square around the screen.
    1. Mark the element as draggable:

        ```
        <div id="red-square" draggable="true"></div>
        ```

7. Given HTML page that has 300 DIVs, create one click event subscription that will print the id of the element clicked on to the console.

    ```
    document.body.addEventListener("click", event => {
        console.log(event.target.id);
    })
    ```

8. Identify the definition of the bubbling principle.
    When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.

JSON

1. Identify and generate valid JSON-formatted strings.
    String in JS: `'this is "text"'`
    String in JSON: `"this is \"text\""`
    - use \n for line breaks
    - keys in JSON objects must be surrounded by " quotes
2. Use `JSON.parse` to deserialize JSON-formatted strings.

    ```
    const str = '[1,"hello, \\"world\\"",3.14,{"id":17}]';
    console.log(JSON.parse(str));
        // prints an array with the following entries:
        //   0: 1
        //   1: "hello, \"world\""
        //   2: 3.14
        //   3: { id: 17 }
    ```

3. Use `JSON.stringify` to serialize JS objects.

    ```
    const array = [1, 'hello, "world"', 3.14, { id: 17 }];
    console.log(JSON.stringify(array));
        // prints [1, "hello, \"world\"", 3.14, {"id":17}]
    ```

4. Correctly identify the definition of "serialize".
    Converting your data into a format that can be sent to another computer.
5. Correctly idenfity the definition of "deserialize".
    Getting a message from another computer and converting that message into usable data.