Session 4 Worksheet: Signup Page Project

Welcome to Session 4! Today you'll build a user signup page and use JavaScript objects and arrays to store and manage user data. The goal is to practice reading input, validating data, storing users, and updating the display. You will work step-by-step, following clear instructions and code organization guides. Each task is bolded for clarity, and some tasks include tips on how to structure your code for best results.

JavaScript Tasks (signup.js)

In this section, you will learn how to:

- Get user data from a form in the HTML
- Validate the inputs (age, email, name)
- Store the data in a JavaScript object
- Store these objects in an array
- Display and manage the user list on the page
- Save the array to a file as `.json` (advanced/optional)

You should arrange your code so that each major feature is in its own function, and use comments to separate the steps. This will make your code easier to read and debug.

Task 1: Get Form Input

When the user submits the form, we need to get the data that the user have saved into the form and save it to some variables. This is your first task.

Task: Select the form and its input fields for name, email, and age using `document.getElementById` or `document.querySelector`. Save the data to some well named variables when the form is submitted.

Task: Check that all fields are filled in. If not, show an error message in the `#error-message` div.

Task 2: Validate input

Now that we have the user input, we need to validate that the user actually has filled in the form correctly. We do this by validating what they have written in the form, using `if` and `string` methods.

Task: Check that the email is valid (contains `@` and a dot `.`) and age is between 1 and 120. Show an error message if validation fails.

You should do this using multiple `if` statements, as this is the simplest way to do it. Check for one thing at a time, i.e. start by checking the user's age in the first if, then their email in the next. Here are two methods you probably will need:

1. `[string].includes(text)`. You can check if a text string contains a specific character or substring by calling `.includes` on it. You can use this to check if the user's email contains an `@`. Read more here

2. In order to alert the user, use the `alert([text])` function. Write what you want to the user in the text of the alert function.

Place your validation code inside the form submit event listener, after you have read the input values.

Reading break

In order for you to finish the last tasks, you will have to read up in how `objects` work in JS. Here is a link: Objects Also, you might need to know more about arrays. Here is a link to where you can read about arrays: Arrays

When you finish reading each of these

Task 3: Create and Store User Objects

Now that you have validated the input, you are ready to create a user object and store it in an array. This is a key step in managing user data in JavaScript.

Task: Use the input values to create a new user object with `name`, `email`, and `age` properties.

Guide: Create the object inside your event listener, after validation. Example:

```
let user = { name: nameValue, email: emailValue, age: ageValue };
```

Task: Log the user object to the console to check your work. This helps you see if your object looks correct.

Task: At the top of your script, create an empty array called `users` to store all user objects:

```
let users = [];
```

Task: When a new user object is created and validated, add it to the `users` array using `push()`:

```
users.push(user);
```

Guide: Keep your array and user creation code at the top of your script, and only add to the array after successful validation.

Task 4: Display Users

Now you want to show all users on the page so you can see who has signed up. This is done by updating the HTML whenever the array changes.

Task: Select the `#user-list` div in your HTML using `document.getElementById` or `document.querySelector`.

Task: Write a function to display all users in the array as a list inside `#user-list`. You can use a loop (a `for ... of` loop is perfect for this) to go through the array and build up the HTML. This means that you have to:

1. Make a new HTML tag for each user
2. Add this tag as a child to the `user-list div` in the html

Use the slides from last session to get inspiration on how to do this.

Task: Call this function every time the array changes (after adding, editing, or removing a user) so the display is always up to date.

Task (Optional): Add sorting by name or age. You can use the `sort()` method on the array to do this.

Guide: Write a separate function called `renderUsers()` that updates the user list. Call this function after any change to the array.

Task 5: Remove and Edit Users (Advanced)

Sometimes you need to remove or edit users. This is a common feature in user management systems.

Task: For each user displayed by the code you made in the previous task, add a "Remove" button next to their details. When clicked, ask the user to confirm before removing.

Task: When the button is clicked, use the following code to find the user's index and remove them from the array:

```
let index = users.findIndex(user => user.email === emailToRemove);
if (index !== -1) {
  users.splice(index, 1);
  renderUsers();
}
```

Replace `emailToRemove` with the email of the user you want to remove. This code will update the display after removing the user. Add this code to your remove button's event handler.

Task: For each user displayed, add an "Edit" button next to their details. When clicked, pre-fill the form with the user's details for editing.

In your `renderUsers()` function, after you create the HTML for each user, also create an "Edit" button. Add an event listener to this button so that when it is clicked, you set the form's input fields (`nameInput`, `emailInput`, `ageInput`) to the values of the selected user.

Task: On form submit, update the user object in the array and update the display.

When the form is submitted after editing, check if you are in "edit mode" (for example, by checking if `editingIndex` is set). If so, update the user object at that index in the `users` array with the new values from the form. Then call `renderUsers()` to update the display. Finally, clear the `editingIndex` variable so the form goes back to normal add mode.

Example code for updating the user:

```
if (editingIndex !== undefined) {
  users[editingIndex] = {
    name: nameInput.value,
    email: emailInput.value,
    age: ageInput.value
  };
  editingIndex = undefined;
  renderUsers();
}
```

Task 6: Search and Filter Users (Advanced)

As your list grows, it helps to be able to search for users by name or email.

Task: Add a search input field above the user list in your HTML (already present).

Task: Select the input in your JS and listen for changes.

Task: On every input change, filter the `users` array to only include users whose name or email matches the search text (use `filter`).

Task: Display the filtered list in the `#user-list` div.

Guide: Write a function called `filterUsers(searchText)` that returns only the users that match the search. Call `renderUsers()` with the filtered array.

Task 7: Extra Features (Optional & Advanced)

If you finish the main tasks, try these extra features to make your app even better!

Task: Clear the form after successful signup so it's ready for the next user.

Task: Show error messages for invalid input in a user-friendly way.

Task: Persist users in localStorage so the list is saved between page reloads. You can use `localStorage.setItem` and `localStorage.getItem` for this.
