Review of JavaScript Program Designs for Suleyman Overload Pro

For the JavaScript integration into my fitness website, *Suleyman Overload Pro*, I have designed and implemented several features aimed at enhancing the user experience and addressing the needs of the personas developed in Unit 1. Below is an overview of each program design, including its purpose, functionality, and relation to the scenarios.

---

1. Toggle “Read More” Feature (Blog Page)

- **Purpose**: The primary goal of this feature is to give users control over the content they want to view. By adding a toggle function to each blog post, users can either expand the post to read more or collapse it to save space.
- **Design**:
    - JavaScript is used to control the visibility of additional text. The toggle button dynamically changes its label between "Read more" and "Read less" based on the content's state.
    - **Objects/Functions**: `toggleReadMore(id)` function; DOM elements such as buttons and paragraphs are targeted.
    - **Variables**: `dots`, `moreText`, `btnText` (all capturing specific sections of the content to be controlled).
- **Program Flow**: When a user clicks the “Read more” button, the JavaScript function hides the dots, reveals the extended content, and changes the button label to "Read less." When clicked again, the content is collapsed.
- **Relation to Personas**: This feature is essential for *Jane*, who likes to browse fitness articles without feeling overwhelmed by long text blocks. It improves usability by giving her control over how much she wants to read.
- **Code:**

```
      /**
       * Toggles the visibility of the full blog post content and updates the read
more/less button.
       *
       * @param {string} id - The unique identifier for the blog post.
       */
      function toggleReadMore(id) {
          // Get the DOM elements for the specific blog post
```

```
        var dots = document.getElementById("dots-" + id);

        var moreText = document.getElementById("more-" + id);

        var btnText = document.getElementById("read-more-btn-" + id);



        // Check if the full content is currently hidden

        if (dots.style.display === "none") {

            // If full content is visible, hide it and show the ellipsis

            dots.style.display = "inline";

            btnText.innerText = "Read more";

            moreText.style.display = "none";

        } else {

            // If full content is hidden, show it and hide the ellipsis

            dots.style.display = "none";

            btnText.innerText = "Read less";

            moreText.style.display = "inline";

        }

    }
```

---

2. Fitness Calculators (BMI, One-Rep Max, and Maintenance Calories)

- **Purpose**: These calculators are designed to offer users immediate feedback based on their personal fitness data. The BMI calculator, one-rep max calculator, and calorie maintenance calculator allow users to calculate important metrics without leaving the page.
- **Design**:

    - Each calculator is embedded in its respective section and responds to user inputs via event handlers (on button click).
    - **Objects/Functions**:
        - `calculateBMI()`: Calculates BMI using height and weight inputs.
        - `calculateOneRepMax()`: Calculates one-rep max using weight lifted and repetitions.
        - `calculateMaintenanceCalories()`: Calculates daily caloric maintenance based on weight, height, age, gender, and activity level.
    - **Variables**: Input fields such as `height`, `weight`, `age`, and dropdowns for gender and activity level. Results are displayed using `innerText`.
- **Program Flow**: Each function captures user inputs, performs the necessary calculations using formulas, and then displays the result below the form.
- **Relation to Personas**: For *John*, a beginner who wants personalized workout recommendations, and *Mark*, an advanced user looking for precise fitness data, these calculators provide essential tools to track progress and set goals.
- **BMI Calculator code:**

```
      /**
       * Calculates the Body Mass Index (BMI) based on user input.
       *
       * This function retrieves the height and weight values entered by the user,
       * validates the input, calculates the BMI using the formula: weight (kg) /
(height (m))^2,
       * and displays the result or an error message if the input is invalid.
       *
       * The BMI categories are also provided for context.
       */
      function calculateBMI() {
          // Retrieve height and weight values from input fields
          const height = parseFloat(document.getElementById('height').value);
          const weight = parseFloat(document.getElementById('weight').value);

          // Check if both height and weight are positive values and not NaN
          if (height > 0 && weight > 0 && !isNaN(height) && !isNaN(weight)) {
              // Calculate BMI
              // Note: Height is converted from cm to m by dividing by 100
              const bmi = (weight / ((height / 100) ** 2)).toFixed(1);

              // Determine BMI category
              let category;
              if (bmi < 18.5) {
                  category = "Underweight";
              } else if (bmi < 25) {
```

```
                category = "Normal weight";
            } else if (bmi < 30) {
                category = "Overweight";
            } else {
                category = "Obese";
            }

            // Display the calculated BMI and category
            document.getElementById('bmi-result').innerHTML = `Your BMI is
${bmi}<br>Category: ${category}`;
        } else {
            // Display an error message if input is invalid
            document.getElementById('bmi-result').innerText = 'Please enter valid
positive numbers for height and weight.';
        }
    }
```

- **One rep max Calculator:**

```
    /**
     * Calculates the One Rep Max (1RM) based on weight lifted and number of reps.
     *
     * @description
     * This function uses the Brzycki formula: 1RM = weight * (36 / (37 - reps))
     * It's considered accurate for rep ranges up to 10.
     *
     * @function
     * @name calculateOneRepMax
     *
     * @returns {void} - Updates the DOM with the calculation result or error
message
     */
    function calculateOneRepMax() {
        // Parse input values from the DOM
        const weight = parseFloat(document.getElementById('weight-lifted').value);
        const reps = parseInt(document.getElementById('reps').value);

        // Validate input and calculate 1RM
        if (weight > 0 && reps > 0 && reps <= 10) {
            // Apply Brzycki formula and round to 2 decimal places
            const oneRepMax = (weight * (36 / (37 - reps))).toFixed(2);
            // Display the calculated 1RM
```

```
            document.getElementById('one-rep-max-result').innerText = `Your
estimated One Rep Max is ${oneRepMax} kg`;
        } else if (weight > 0 && reps > 10) {
            // Warn user if reps exceed recommended range for accurate calculation
            document.getElementById('one-rep-max-result').innerText = 'For accurate
results, please enter 10 reps or fewer.';
        } else {
            // Display error message for invalid input
            document.getElementById('one-rep-max-result').innerText = 'Please enter
valid positive numbers for weight and reps.';
        }
    }
```

- **Calorie Maintainence Calculator:**

```
function calculateMaintenanceCalories
            // Parse input values from the DOM
            const weight =
parseFloat(document.getElementById('weight-maint').value);
            const height =
parseFloat(document.getElementById('height-maint').value);
            const age = parseInt(document.getElementById('age').value);
            const gender = document.getElementById('gender').value;
            const activityLevel =
parseFloat(document.getElementById('activity-level').value);

            // Validate input
            if (weight > 0 && height > 0 && age > 0 && !isNaN(activityLevel)) {
                // Calculate BMR using Mifflin-St Jeor Equation
                let bmr;
                if (gender === "male") {
                    // BMR formula for males
                    bmr = (10 * weight) + (6.25 * height) - (5 * age) + 5;
                } else if (gender === "female") {
                    // BMR formula for females
                    bmr = (10 * weight) + (6.25 * height) - (5 * age) - 161;
                } else {
                    // Handle invalid gender input
```

```
document.getElementById('maintenance-calories-result').innerText = 'Please
select a valid gender.';
                    return;
                }


                // Calculate maintenance calories by multiplying BMR with
activity level
                const maintenanceCalories = (bmr * activityLevel).toFixed(0);


                // Display the calculated maintenance calories
                document.getElementById('maintenance-calories-result').innerText
= `Your estimated Maintenance Calories are ${maintenanceCalories} kcal/day`;
            } else {
                // Display error message for invalid input
                document.getElementById('maintenance-calories-result').innerText
= 'Please enter valid positive numbers for all fields.';
            }
        }
```

---

3. Form Validation (Contact Page)

- **Purpose**: The goal of this design is to ensure that users submit valid data when contacting the site. It prevents incomplete or incorrect data entry, reducing potential errors and frustration for users.
- **Design**:
    - JavaScript checks for empty fields and verifies that the email is in the correct format using a regular expression pattern.
    - **Objects/Functions**: `validateForm()` function, which returns `false` if the data is invalid and prevents form submission.
    - **Variables**: Input fields for name, email, and message. Email validation is performed using the regular expression stored in the `emailPattern` variable.

- **Program Flow**: When the user submits the form, the JavaScript function checks each field for missing data and validates the email format. If any field is invalid, an alert message is displayed, and the form is not submitted.
- **Relation to Personas**: For users like *Sarah*, who may be using the contact form for inquiries or fitness advice, this feature ensures that their messages are correctly formatted and prevents errors that could lead to incomplete submissions.
- **Code:**

```
/**
 * Validates the contact form before submission.
 * Checks for empty fields and valid email format.
 * @returns {boolean} True if form is valid, false otherwise.
 *
 * Sample tests (run these in the browser console):
 *
 * // Test empty fields
 * document.forms["contactForm"]["name"].value = "";
 * document.forms["contactForm"]["email"].value = "";
 * document.forms["contactForm"]["message"].value = "";
 * console.log(validateForm()); // Should return false
 *
 * // Test invalid email
 * document.forms["contactForm"]["name"].value = "John Doe";
 * document.forms["contactForm"]["email"].value = "invalid.email";
 * document.forms["contactForm"]["message"].value = "Test message";
 * console.log(validateForm()); // Should return false
 *
 * // Test valid input
```

```
 * document.forms["contactForm"]["name"].value = "John Doe";
 * document.forms["contactForm"]["email"].value = "john@example.com";
 * document.forms["contactForm"]["message"].value = "Test message";
 * console.log(validateForm()); // Should return true
 */
function validateForm() {
    // Get form field values
    const name = document.forms["contactForm"]["name"].value.trim();
    const email = document.forms["contactForm"]["email"].value.trim();
    const message = document.forms["contactForm"]["message"].value.trim();

    // Check for empty fields
    if (!name || !email || !message) {
        alert("All fields must be filled out");
        return false; // Prevent form submission
    }

    // Validate email format using a regular expression
    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailPattern.test(email)) {
        alert("Please enter a valid email address");
        return false; // Prevent form submission
    }
```

```
        // All validations passed

        return true;

    }
```

---

4. Time-Sensitive Greeting (Home Page)

- **Purpose**: This feature creates a welcoming, personalized touch by displaying a different greeting message based on the time of day. This simple JavaScript functionality enhances the user experience by making the site feel more dynamic and responsive.
- **Design**:
    - The script checks the current time when the page loads and assigns a greeting (morning, afternoon, or evening) accordingly.
    - **Objects/Functions**: The `window.onload` event triggers a function that uses the `Date()` object to retrieve the current hour.
    - **Variables**: `hour`, `greeting`, `welcome-message` (used to display the greeting text).
- **Program Flow**: Once the page loads, the JavaScript function determines the current hour and updates the content of the `welcome-message` element with an appropriate greeting based on the time of day.
- **Relation to Personas**: This feature subtly improves the experience for all users, especially returning visitors like *John*, by creating a friendly, time-sensitive greeting that adds a human touch to the site.

- **Code:**

```
<script>

    /**

     * This script sets a personalized greeting based on the time of day.

     * It runs when the window loads, determining the appropriate greeting

     * (morning, afternoon, or evening) and updates the welcome message.

     *

     * The script performs the following steps:
```

```
 * 1. Waits for the window to fully load.
 * 2. Gets the current hour using the Date object.
 * 3. Determines the appropriate greeting based on the hour.
 * 4. Finds the welcome message element in the DOM.
 * 5. Updates the welcome message with the greeting and introduction.
 *
 * Error handling is included to log an error if the welcome message
 * element is not found in the DOM.
 */
window.onload = function() {
    // Initialize greeting variable
    let greeting;
    // Get current hour (0-23)
    const hour = new Date().getHours();


    // Set greeting based on time of day
    if (hour < 12) {
        greeting = "Good Morning";
    } else if (hour < 18) {
        greeting = "Good Afternoon";
    } else {
        greeting = "Good Evening";
    }
```

```
            // Find the welcome message element
            const welcomeMessage = document.getElementById("welcome-message");



            // Update the welcome message with the appropriate greeting
            if (welcomeMessage) {
                welcomeMessage.textContent = `${greeting}, My name is Suleyman Kiani.
Welcome to my fitness journey!`;
            } else {
                // Log an error if the element is not found
                console.error("Element with id 'welcome-message' not found");
            }
        };
    </script>
```

Design Considerations

- **Flow Diagrams/Pseudo-Code**: Each of the program designs follows a clear flow:
    - User input is captured.
    - JavaScript performs the required logic (calculations, validation, toggling).
    - The result is displayed on the page or an action is triggered (e.g., preventing form submission).
- **Integration with HTML/CSS**: All JavaScript is well-integrated with the HTML structure and adheres to the design language of the website. The calculators, validation forms, and interactive buttons are consistent with the CSS styling, maintaining a seamless user experience.
- **Scope and Complexity**: Each design is crafted to balance functionality with ease of use. While the calculators involve basic mathematical logic, the form validation uses regular expressions, offering a higher level of technical complexity.

Conclusion

The JavaScript program designs were carefully crafted to fit the needs of my personas and improve user experience across different sections of the website. From interactive calculators to form validation and dynamic content, each feature aligns with the goals of the site and provides valuable functionality for users at various stages of their fitness journey. These designs enhance usability, accessibility, and interactivity, ensuring that the website remains engaging and user-friendly for a wide audience.