# AI ASSIGNMENT:14.1

HTNO:2403A51284

BATCH:12

Task 1 – Portfolio Website Design
You are building a personal portfolio website to showcase your work.
Requirements:
• Create sections for About Me, Projects, and Contact.
• Use AI to:
o Suggest color palettes and typography.
o Create a responsive layout with Grid/Flexbox.
o Add smooth scrolling navigation.

PROMPT:

Build a responsive personal portfolio website using HTML, CSS, and JavaScript.
Requirements:

1. Sections: About Me, Projects, and Contact.
2. Use CSS Grid or Flexbox for a clean, responsive layout that works on desktop and mobile.
3. Suggest a professional color palette (3–4 colors) and suitable typography (heading + body font).
4. Add smooth scrolling navigation with links in a top navbar that jump to each section.
5. Keep the design modern and minimal, with hover effects for buttons/links.
6. Include placeholder text and example projects so I can replace them later.
7. Write clean, well-commented code."*

CODE:

```
[5]
0s
css_smooth_scrolling = """
html {
    scroll-behavior: smooth;
}
"""

# Combine all CSS
full_css = css_styling + css_smooth_scrolling

print("CSS for Smooth Scrolling:")
print(css_smooth_scrolling)
```

OUTPUT:

```
CSS for Smooth Scrolling:

html {
    scroll-behavior: smooth;
}
```

Combine and Display

Variables Terminal Toggle Gemini 3:23PM Python 3

Task 2 – Online Store Product Page
Design a product display page for an online store.
Requirements:
• Display product image, title, price, and "Add to Cart" button.
• Use AI to:
o Style with BEM methodology.
o Make layout responsive.
o Add hover effects and "Add to Cart" alert

PROMPT:

*"Build a responsive product display page for an online store using HTML, CSS, and JavaScript.
Requirements:

1. Show product image, product title, price, and an “Add to Cart” button.
2. Style the page using the **BEM methodology** for class naming.
3. Make the layout responsive with Flexbox/Grid so it looks good on desktop and mobile.
4. Add hover effects on the product card and button.
5. When the user clicks 'Add to Cart,' display a JavaScript alert saying 'Product added to cart!'.
6. Use clean, minimal styling with comments in code for clarity."*

CODE:

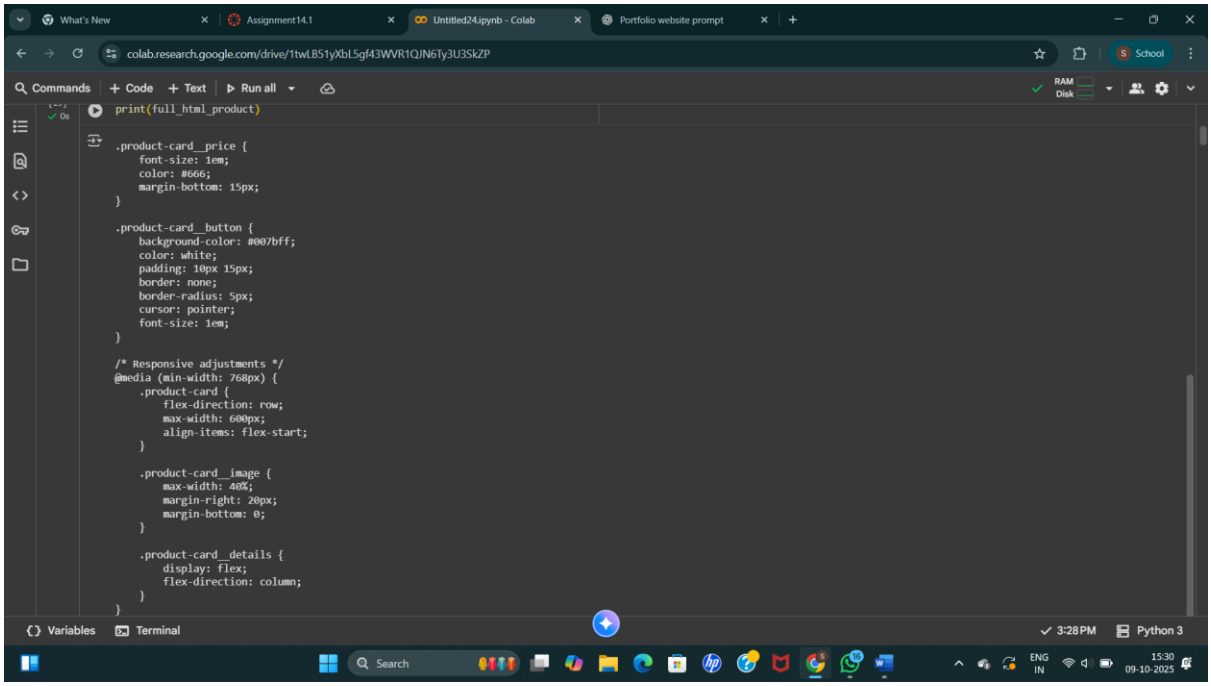
**Reasoning:** Combine the generated HTML structure, CSS styling, and JavaScript code into a single HTML file and then display its content.

```python
full_html_product = f"""
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Product Display</title>
    <style>
        {full_css}
    </style>
</head>
<body>
    <div class="product-card">
        <img src="placeholder.jpg" alt="Product Image" class="product-card__image">
        <h2 class="product-card__title">Product Title</h2>
        <p class="product-card__price">$19.99</p>
        <button class="product-card__button">Add to Cart</button>
    </div>

    <script>
        {js_add_to_cart_alert}
    </script>
</body>
</html>
"""

with open("product_display.html", "w") as f:
    f.write(full_html_product)

print("Generated product_display.html:")
print(full_html_product)
```

OUTPUT:

```python
print(full_html_product)
```

```
Generated product_display.html:

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Product Display</title>
    <style>

.product-card {
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 8px;
    max-width: 300px;
    margin: 20px auto;
}

.product-card__image {
    max-width: 100%;
    height: auto;
    margin-bottom: 15px;
}

.product-card__title {
    font-size: 1.2em;
    margin-bottom: 10px;
    color: #333;
}

.product-card__price {
    font-size: 1em;
    color: #666;
```




Task 3 – Event Registration Form
Build an event registration form for a conference.
Requirements:
• Collect name, email, phone number, and session selection.
• Use AI to:
o Add form validation with JavaScript.
o Make the form accessible with labels and ARIA.
o Style with a professional look.

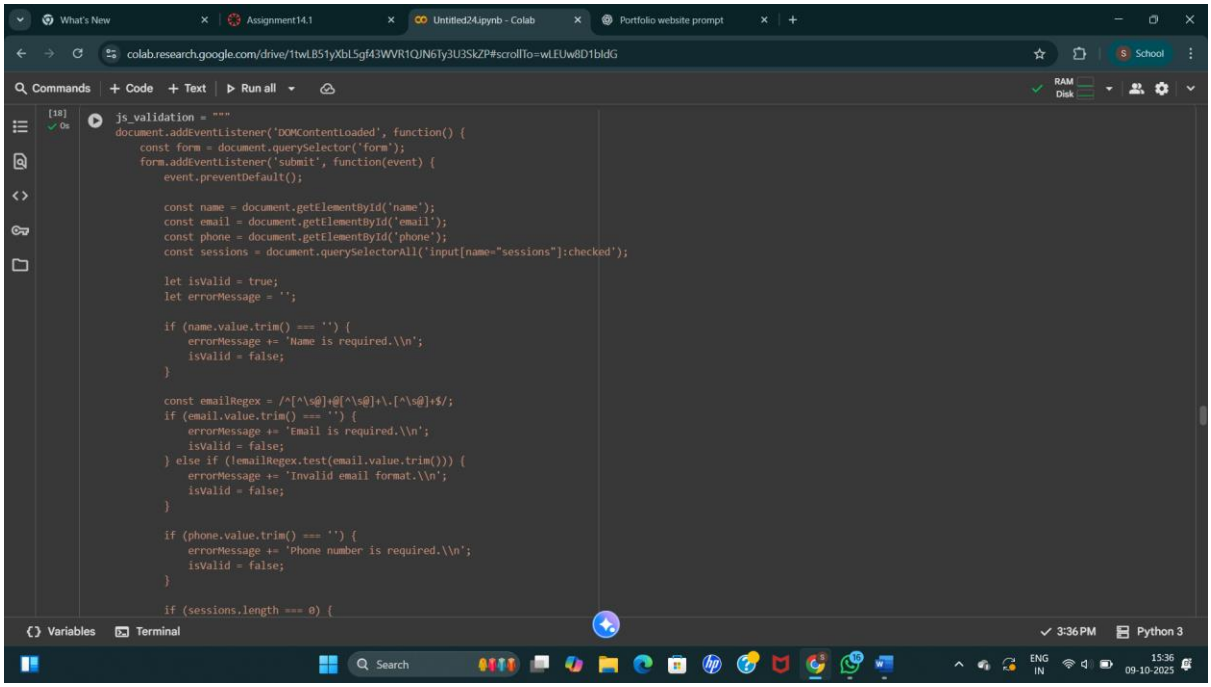
PROMPT:

"Build a responsive event registration form for a conference using HTML, CSS, and JavaScript.
Requirements:

1. Collect user details: name, email, phone number, and session selection (dropdown or radio buttons).
2. Add **form validation** with JavaScript (check required fields, validate email format, and phone number length).
3. Ensure the form is **accessible**: include proper <label> tags, use ARIA attributes where needed, and maintain good contrast.
4. Style the form with a clean, **professional look** (centered layout, padding, consistent typography, and button styling).

5. Show inline error messages if validation fails, and a success message on valid submission.
6. Write clean, well-commented code."*

CODE:

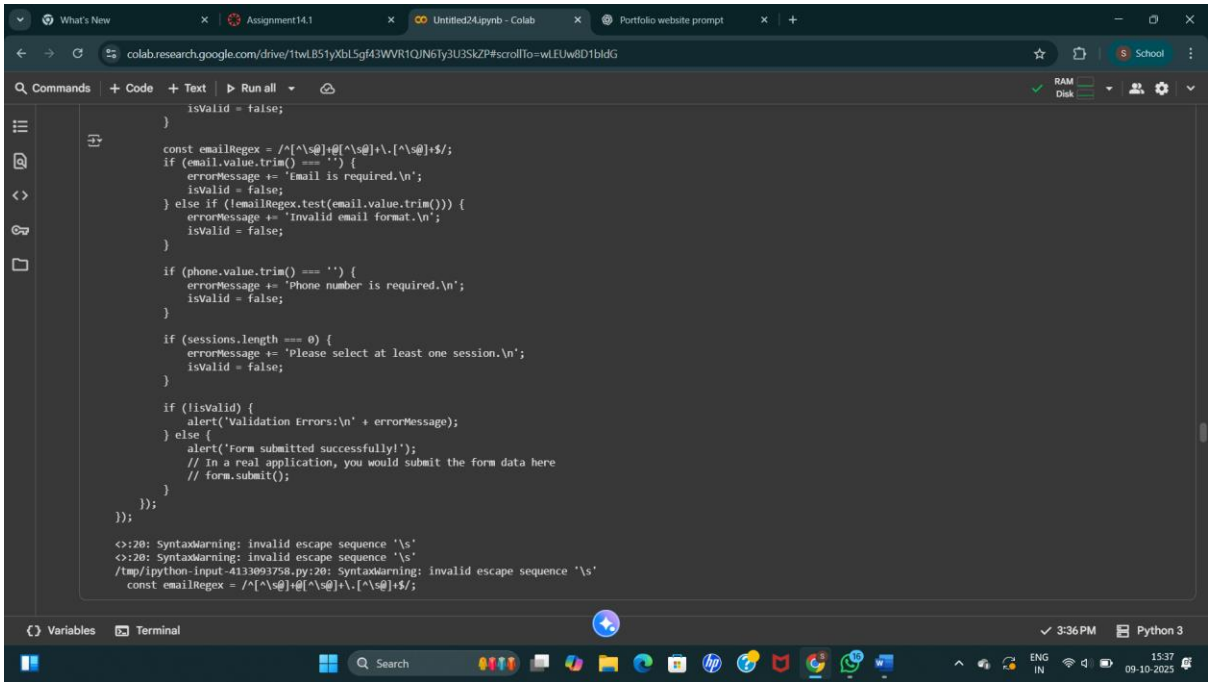


```
js_validation = """
document.addEventListener('DOMContentLoaded', function() {
    const form = document.querySelector('form');
    form.addEventListener('submit', function(event) {
        event.preventDefault();

        const name = document.getElementById('name');
        const email = document.getElementById('email');
        const phone = document.getElementById('phone');
        const sessions = document.querySelectorAll('input[name="sessions"]:checked');

        let isValid = true;
        let errorMessage = '';

        if (name.value.trim() === '') {
            errorMessage += 'Name is required.\\n';
            isValid = false;
        }

        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (email.value.trim() === '') {
            errorMessage += 'Email is required.\\n';
            isValid = false;
        } else if (!emailRegex.test(email.value.trim())) {
            errorMessage += 'Invalid email format.\\n';
            isValid = false;
        }

        if (phone.value.trim() === '') {
            errorMessage += 'Phone number is required.\\n';
            isValid = false;
        }

        if (sessions.length === 0) {
```

```
        }

        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (email.value.trim() === '') {
            errorMessage += 'Email is required.\\n';
            isValid = false;
        } else if (!emailRegex.test(email.value.trim())) {
            errorMessage += 'Invalid email format.\\n';
            isValid = false;
        }

        if (phone.value.trim() === '') {
            errorMessage += 'Phone number is required.\\n';
            isValid = false;
        }

        if (sessions.length === 0) {
            errorMessage += 'Please select at least one session.\\n';
            isValid = false;
        }

        if (!isValid) {
            alert('Validation Errors:\\n' + errorMessage);
        } else {
            alert('Form submitted successfully!');
            // In a real application, you would submit the form data here
            // form.submit();
        }
    });
});
"""

print(js_validation)
```

OUTPUT:

```
document.addEventListener('DOMContentLoaded', function() {
    const form = document.querySelector('form');
    form.addEventListener('submit', function(event) {
        event.preventDefault();

        const name = document.getElementById('name');
        const email = document.getElementById('email');
        const phone = document.getElementById('phone');
        const sessions = document.querySelectorAll('input[name="sessions"]:checked');

        let isValid = true;
        let errorMessage = '';

        if (name.value.trim() === '') {
            errorMessage += 'Name is required.\n';
            isValid = false;
        }

        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (email.value.trim() === '') {
            errorMessage += 'Email is required.\n';
            isValid = false;
        } else if (!emailRegex.test(email.value.trim())) {
            errorMessage += 'Invalid email format.\n';
            isValid = false;
        }

        if (phone.value.trim() === '') {
            errorMessage += 'Phone number is required.\n';
            isValid = false;
        }

        if (sessions.length === 0) {
            errorMessage += 'Please select at least one session.\n';
            isValid = false;
        }
```




Task Description #4 (Data – Fetch API & Render List with Loading/Error States)

• Task: Fetch JSON from an API and render items to the DOM with loading and error UI.

• Instructions:

o Ask AI to write fetch() logic, create DOM nodes safely, and add skeleton/loading text. Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.

PROMPT:

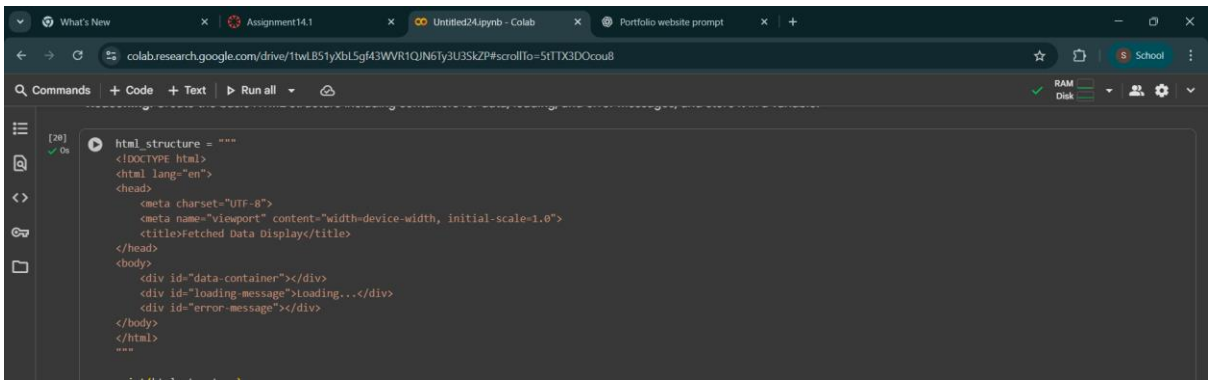
"Build a JavaScript program that fetches JSON data from a public API (e.g., https://jsonplaceholder.typicode.com/posts) and renders the items into the DOM.

Requirements:

1. Use fetch() with proper error handling.
2. Show a **loading state** (skeleton or loading text) before data arrives.

3. If the request fails, display an **error message** in the DOM.
4. Create DOM nodes **safely** (no innerHTML injection).
5. Write **at least 3 assert-based test cases** (using simple JS console.assert) to validate:
    - Data is fetched successfully.
    - Loading state is shown before rendering.
    - Error message appears when fetch fails.
6. Generate initial code with inline comments, then analyze whether tests pass.
7. Improve the code with best practices and accessibility (ARIA live regions for loading/error states).
8. Provide inline explanations and clean formatting.
9. Finally, compile everything into a short **report (Word/PDF)** including: prompts, test cases, code versions, outputs/screenshots, and analysis."*

CODE:




OUTPUT: