

Assignment-14.3

NAME : MERUGU RAKSHITH

ENROLLMENT NO : 2403A52007

BATCH NO: 02

SUBJECT: AI ASSISTANT CODING

Task Description #1 – AI-generated HTML Page

Task: Ask AI to generate a simple HTML homepage for a "Student Info Portal" with a header, navigation menu, and footer.

Prompt

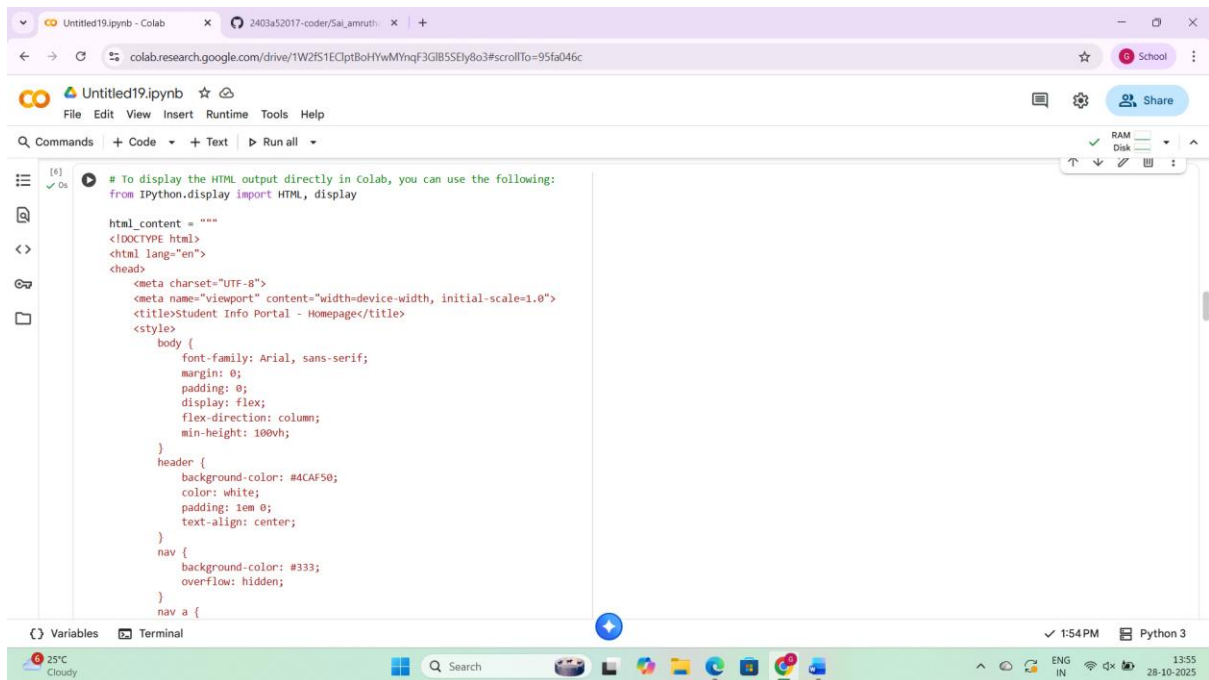
Generate a simple HTML homepage for a "**Student Info Portal**".

The page should include a clear **header**, a **navigation menu** (e.g., Home, About, Contact), a **main content section**, and a **footer**.

Make the structure clean, readable, and well-formatted with proper HTML5 tags and indentation.

Also, explain how each section contributes to the overall webpage layout.

Code and output

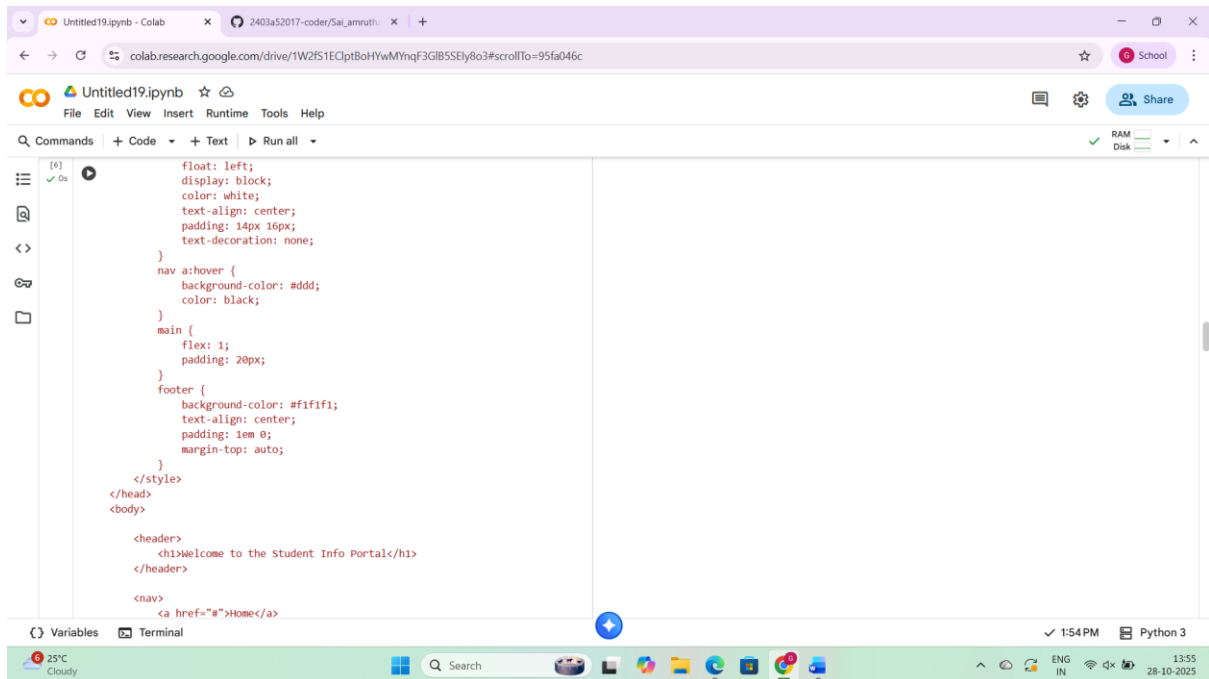


The screenshot shows a Google Colab notebook titled 'Untitled19.ipynb'. The first cell contains a comment and a code snippet for displaying HTML output. The code defines a variable 'html_content' with a string containing HTML and CSS for a 'Student Info Portal'.

```
# To display the HTML output directly in colab, you can use the following:
from IPython.display import HTML, display

html_content = """
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Info Portal - Homepage</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      min-height: 100vh;
    }
    header {
      background-color: #4CAF50;
      color: white;
      padding: 1em 0;
      text-align: center;
    }
    nav {
      background-color: #333;
      overflow: hidden;
    }
    nav a {

```



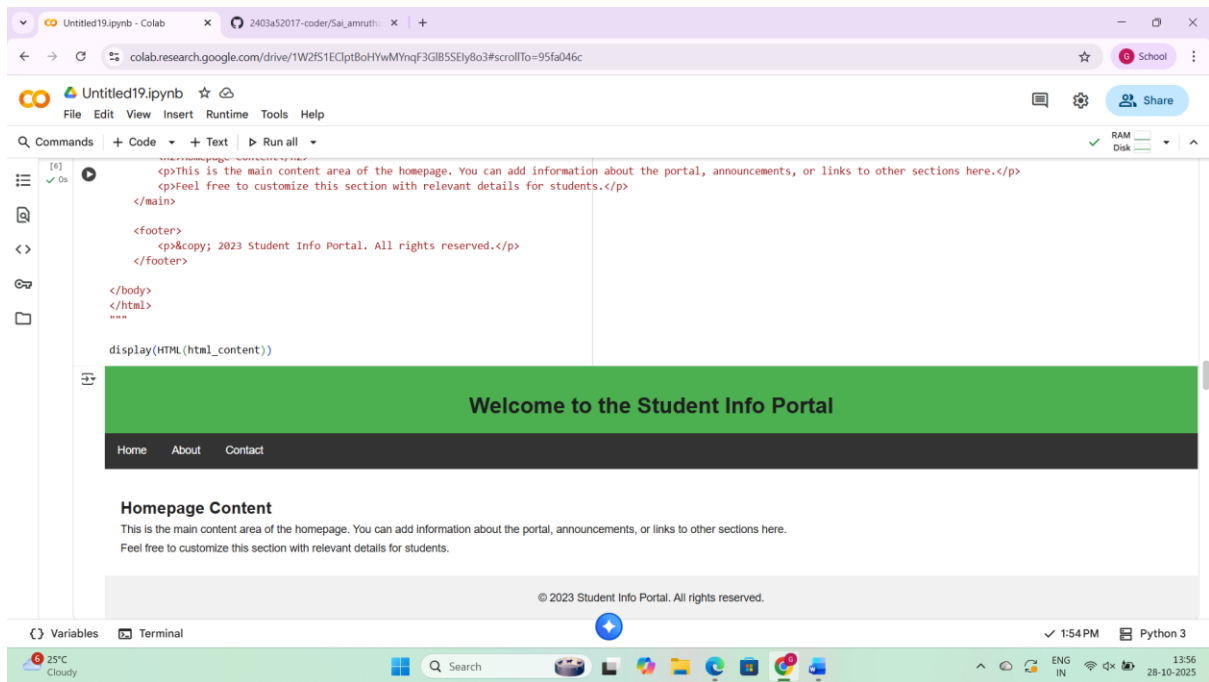
The screenshot shows the second cell of the Google Colab notebook. It continues the CSS and HTML code from the first cell, defining styles for the navigation bar, main content area, and footer, and then closing the HTML document tags.

```
float: left;
display: block;
color: white;
text-align: center;
padding: 14px 16px;
text-decoration: none;
}
nav a:hover {
  background-color: #ddd;
  color: black;
}
main {
  flex: 1;
  padding: 20px;
}
footer {
  background-color: #f1f1f1;
  text-align: center;
  padding: 1em 0;
  margin-top: auto;
}
</style>
</head>
<body>

<header>
  <h1>Welcome to the Student Info Portal</h1>
</header>

<nav>
  <a href="#">Home</a>

```



Observations

- **Semantic HTML5:** The code effectively uses semantic HTML5 tags like `<header>`, `<nav>`, `<main>`, and `<footer>`. This makes the structure of the page clear and helps search engines and assistive technologies understand the content better.
- **Basic Structure:** The code establishes a clear and common webpage layout with distinct sections for the header, navigation, main content, and footer. This is a good foundation for most websites.
- **Internal CSS:** The CSS is embedded directly within the `<style>` tags in the `<head>`. While this is acceptable for small examples, for larger projects, it's generally better to link to an external CSS file for better organization and reusability.
- **Basic Styling:** The CSS provides basic styling for layout and appearance, including setting font family, handling margins and padding, and styling the header and navigation.
- **Responsive Design Meta Tag:** The `<meta name="viewport"...>` tag is included, which is essential for making the webpage responsive and ensuring it displays correctly on different devices.
- **Clear Navigation:** The navigation menu uses a simple list of links within the `<nav>` tag, which is a standard approach.

Task Description #2 – CSS Styling

Task:

Use AI to add CSS styling to Task #1 homepage for:

- Responsive navigation bar.
- Centered content section.
- Footer with light gray background.

Prompt

Add CSS styling to the "Student Info Portal" homepage created in Task #1. The page should include:

- A **responsive navigation bar** that adjusts for different screen sizes.
- A **centered main content section** for better visual balance.
- A **footer with a light gray background** and proper padding.

Make the **CSS clean, readable, and either embedded within the HTML or placed in a <style> block**.

Also, **explain how each CSS rule improves the webpage's appearance and responsiveness**.

Code and output

The screenshot shows a Google Colab notebook with the following content:

```
HTML structure:
1. <!DOCTYPE html> : Declares the document type to be HTML5.
2. <html lang="en"> : The root element of the HTML page. The lang="en" attribute specifies the language of the document.
3. <head> : Contains meta-information about the HTML document, such as:
   o <meta charset="UTF-8"> : Specifies the character encoding for the document, ensuring proper display of various characters.
   o <meta name="viewport" content="width=device-width, initial-scale=1.0"> : Configures the viewport for responsive web design, making the page display well on different devices.
   o <title>Student Info Portal - Homepage</title> : Sets the title that appears in the browser's title bar or tab.
   o <style> : Contains basic CSS rules to style the elements and provide a simple layout.
4. <body> : Contains the visible content of the HTML page.
   o <header> : Represents the introductory content or a container for a set of navigational links. In this case, it holds the main heading of the page.
   o <nav> : Represents a section of a page that contains navigation links. Here, it holds the links for "Home," "About," and "Contact."
   o <main> : Represents the dominant content of the <body>, excluding content that is repeated across multiple pages (like headers, footers, or navigation). This is where the main content of the homepage is placed.
   o <footer> : Represents a footer for its nearest sectioning content or the root element. In this case, it contains copyright information for the entire page.

CSS Styling and Responsiveness:
The <style> block contains CSS rules that enhance the webpage's appearance and responsiveness:
```

The bottom of the notebook shows a terminal window with the command `Toggle Gemini` and a status bar indicating the time is 1:59 PM and the Python version is 3.

Observation

- **Responsive Design Implementation:** The use of the viewport meta tag and the @media query for the navigation bar effectively implements a basic responsive design. The navigation shifts from horizontal to vertical on smaller screens, which is a common pattern for mobile-friendly websites.

- **Flexbox for Layout:** Using `display: flex`, `flex-direction: column`, and `min-height: 100vh` on the body and `flex: 1` on the main content is a modern and effective way to create a sticky footer layout where the footer stays at the bottom of the viewport even if the main content is short.
- **Centering Content:** The `max-width` and `margin: 20px auto` on the main content effectively centers the content block on wider screens and provides some padding around it.
- **Clear Visual Hierarchy:** The header and footer have distinct background colors and padding, which helps create a clear visual hierarchy and separates these sections from the main content.
- **Navigation Styling:** The styling for the navigation links provides good visual feedback on hover and ensures they are easily clickable blocks.
- **Shadow for Depth:** The subtle box-shadow on the main content adds a touch of visual depth and helps it stand out from the background.

The combination of semantic HTML and the applied CSS creates a visually appealing and functional basic homepage that adapts reasonably well to different screen sizes.

Task Description #3 – JavaScript Interactivity

Task: Prompt AI to generate a JS script that validates a simple login form (non-empty username/password).

Prompt

Generate a **JavaScript script** that adds interactivity to a simple **login form** by validating user input.

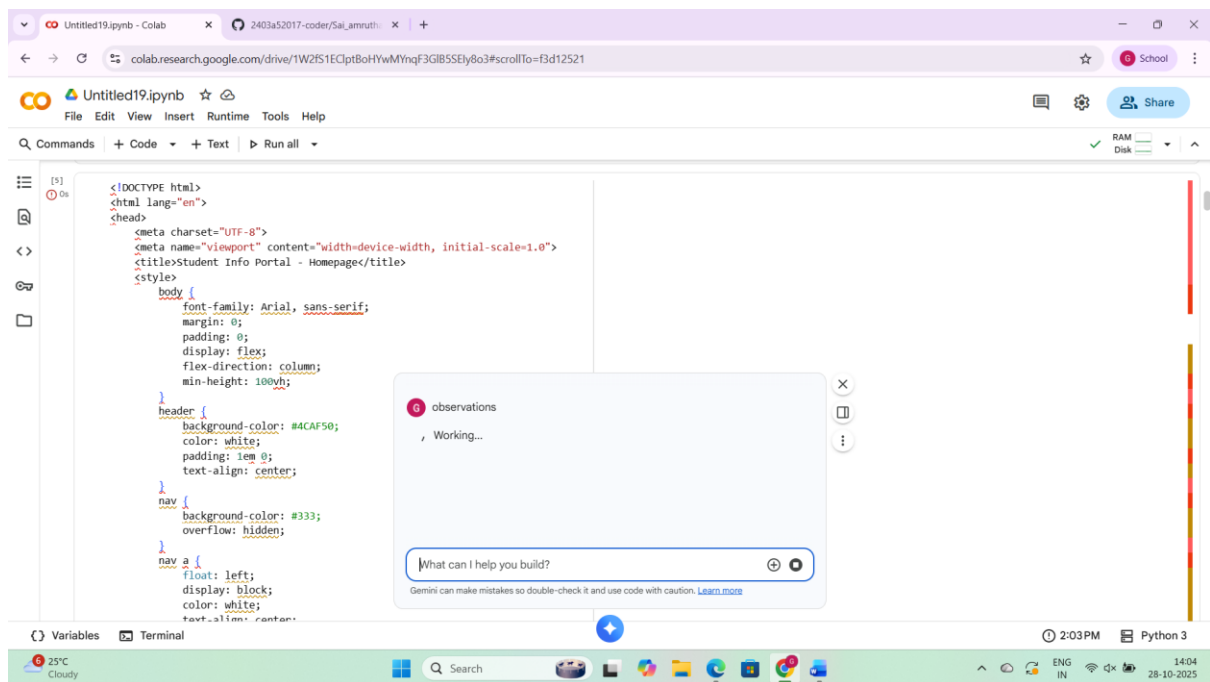
The script should ensure that both **username** and **password** fields are not empty before submission.

Display an **alert or message** if any field is left blank.

Make the code clean, readable, and properly commented.

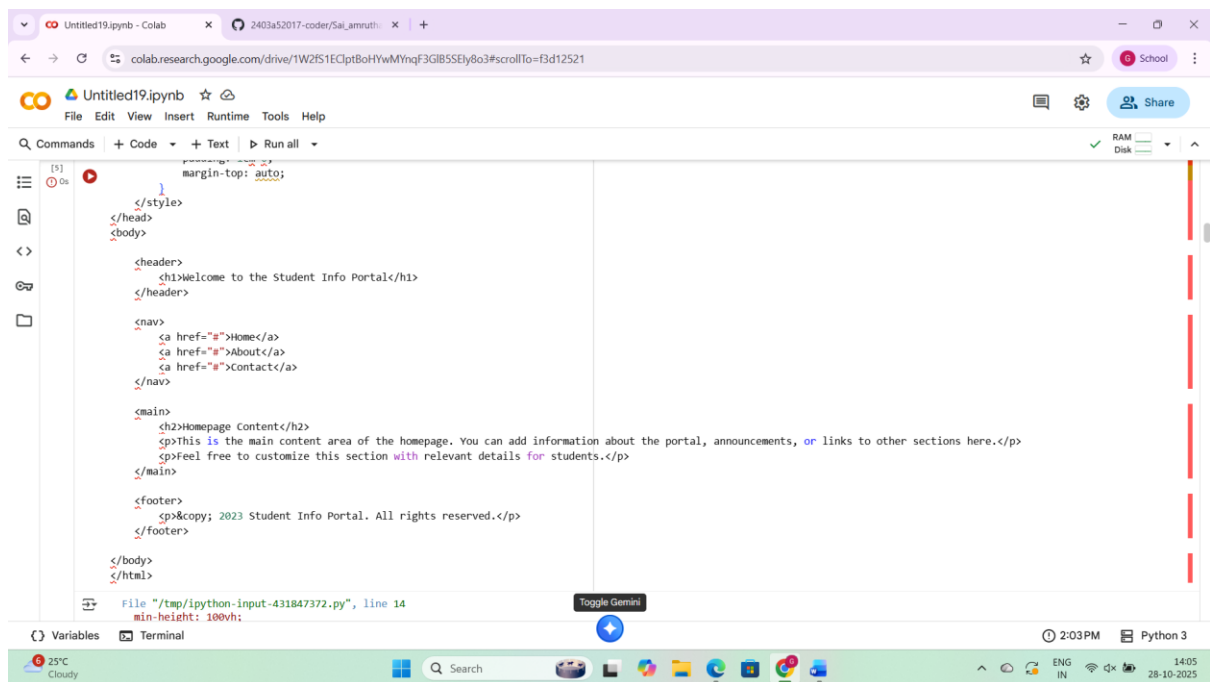
Also, **explain how the JavaScript improves form interactivity and user experience.**

Code and output



The screenshot shows a Google Colab notebook titled 'Untitled19.ipynb'. The code cell contains CSS for a 'Student Info Portal' homepage. A Gemini chat window is open, showing the prompt 'What can I help you build?' and the response 'Working...'. The chat window also includes a disclaimer: 'Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)'.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Info Portal - Homepage</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      min-height: 100vh;
    }
    header {
      background-color: #4CAF50;
      color: white;
      padding: 1em 0;
      text-align: center;
    }
    nav {
      background-color: #333;
      overflow: hidden;
    }
    nav a {
      float: left;
      display: block;
      color: white;
      text-align: center;
    }
  </style>
</head>
<body>
  <header>
    <h1>Welcome to the Student Info Portal</h1>
  </header>
  <nav>
    <a href="#">Home</a>
    <a href="#">About</a>
    <a href="#">Contact</a>
  </nav>
  <main>
    <h2>Homepage Content</h2>
    <p>This is the main content area of the homepage. You can add information about the portal, announcements, or links to other sections here.</p>
    <p>Feel free to customize this section with relevant details for students.</p>
  </main>
  <footer>
    <p>&copy; 2023 Student Info Portal. All rights reserved.</p>
  </footer>
</body>
</html>
```



The screenshot shows the same Google Colab notebook with the completed HTML code. The code includes the CSS from the previous screenshot and adds the HTML structure for the homepage, including a header, navigation menu, main content area, and footer. A 'Toggle Gemini' button is visible at the bottom of the code cell.

```
margin-top: auto;
</style>
</head>
<body>
  <header>
    <h1>Welcome to the Student Info Portal</h1>
  </header>
  <nav>
    <a href="#">Home</a>
    <a href="#">About</a>
    <a href="#">Contact</a>
  </nav>
  <main>
    <h2>Homepage Content</h2>
    <p>This is the main content area of the homepage. You can add information about the portal, announcements, or links to other sections here.</p>
    <p>Feel free to customize this section with relevant details for students.</p>
  </main>
  <footer>
    <p>&copy; 2023 Student Info Portal. All rights reserved.</p>
  </footer>
</body>
</html>
```

Observation

- It defines a function `calculate_area` that calculates the area of different shapes using a dictionary of lambda functions.
- It includes error handling for unsupported shapes and missing dimensions for rectangles.
- It uses `math.pi` for a more accurate circle area calculation.

- Example usage demonstrates how to call the function for different shapes, including cases with errors.

For the HTML code in cell 95fa046c:

- It defines a basic HTML structure for a "Student Info Portal" homepage with a header, navigation, main content, and footer.
- It includes embedded CSS within the <style> tags for basic styling and layout.
- It uses semantic HTML5 tags like <header>, <nav>, <main>, and <footer>.
- It includes a viewport meta tag for responsiveness.

Task Description #4 – Python Backend Integration

Task: Ask AI to generate a Flask app that serves the HTML form (Task #3) and prints the username on successful login.

Prompt

Generate a flask app that serves html from task 3 and print the username on successful login

Code and output

The screenshot shows a Google Colab notebook interface. At the top, there's a browser window showing a Google Drive link. Below it, the Colab notebook editor is open. The first cell contains JavaScript code with a syntax error: `console.error("Form with ID 'loginForm' not found.");`. The error message is: `SyntaxError: unterminated string literal (detected at line 29)`. Below the error, there's a "Next steps: Explain error" button. The second cell is a subtask prompt: "Create a Python script for a basic Flask application." It includes reasoning: "The subtask requires creating a basic Flask application script by importing Flask, instantiating the app, and setting it up to run in debug mode. This can be done in a single code block." Below the prompt, there's a code block generated by Gemini:

```
from flask import Flask

app = Flask(__name__)

if __name__ == '__main__':
    app.run(debug=True)
```

 The bottom status bar shows "Executing (1m 44s)" and "Python 3".

Observation

- This code appears to be JavaScript intended for client-side form validation.
- It attempts to get form and input elements by their IDs.

- It adds an event listener for the form's submit event to prevent default submission.
- It checks if the username and password fields are empty after trimming whitespace.
- It uses `alert` and `console.log` for feedback, which are typical in browser-based JavaScript.
- The error you encountered earlier in this cell (`SyntaxError: unterminated string literal`) was because JavaScript was being run in a Python environment.

For the code in cell a930d1ce:

- This is a basic Python script for a Flask web application.
- It imports the Flask class.
- It creates an instance of the Flask application.
- The `if __name__ == '__main__':` block ensures that the development server runs only when the script is executed directly.
- `app.run(debug=True)` starts the development server in debug mode, which is useful for development as it provides detailed error pages and reloads automatically on code changes.