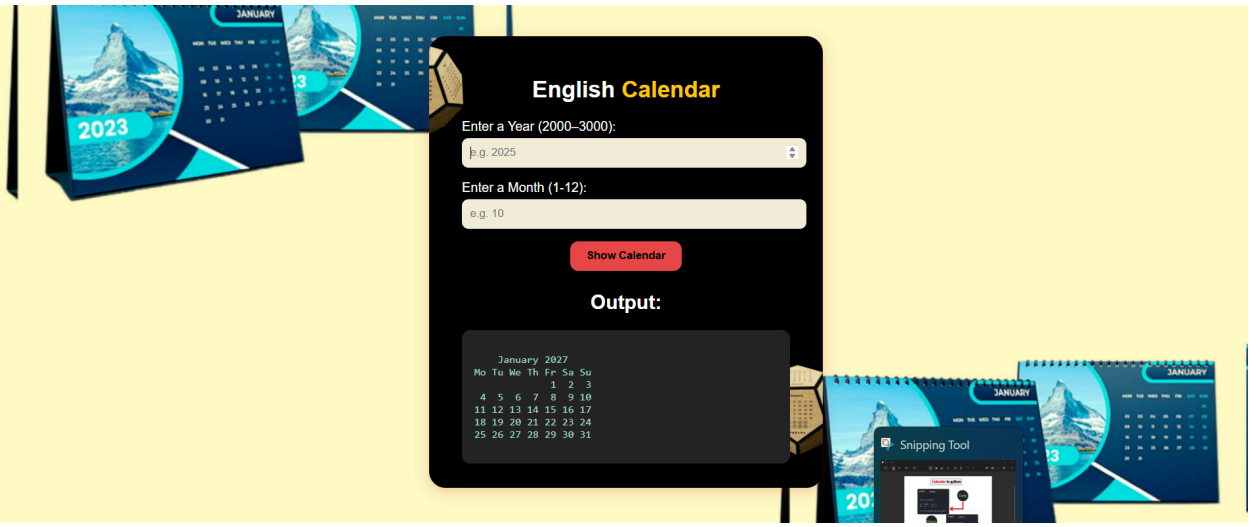

Calendar Project Explanation Note

The project is mainly built using Python's built-in **calendar** module

app.py

```
from flask import Flask, render_template, request
import calendar
```

1. `Flask` → to create the main web application.
2. `render_template` → used to load and display an HTML file from the `templates` folder.

Note:
By default, Flask expects an HTML folder named `templates`, but you can tell Flask to use a different folder when creating the app, like this:

```
app = Flask(name, template_folder='my_folder')  # custom folder name
```

After that, you still **use** `render_template` to load HTML files the function name does **not** change.
3. `request` → used to access data sent by the user (like form input).
4. `import calendar` imports Python's built-in `calendar` module (used to generate the month layout).

```
app = Flask(__name__)
```

1. Creates the **Flask app object**.
2. `__name__` tells Flask where to find files and resources (like templates, static files, etc.).
3. `app` now becomes your main web application that will handle routes and run the server.

```
@app.route('/', methods=['GET', 'POST'])
```

1. This is a **route decorator** that tells Flask which URL should trigger this function. For example:

```
@app.route('/')
def home():
return "Welcome to the Home Page!"
```

- `@app.route('/')` → defines the **URL path** (`/` means the homepage).
- `home()` → is the function that runs when someone visits the homepage.
- `return "..."` → sends text (or HTML) as the response.

```
@app.route('/about')
def about():
return "This is the About Page."
```

`/about` → about page

2. `'/'` means the **home page** (the root URL of your website).
3. `methods=['GET', 'POST']` means the route can handle both:

- **GET** → when you first open the page
- **POST** → when you submit the form (with year and month)

```
def index():
```

Defines a function named `index()` that will run when someone visits `/`.

```
cal_output = None
year = month = None
```

- Initializes three variables:
  - `cal_output` will store the generated calendar text.
  - `year` and `month` will store user input from the form.
- Initially all set to `None` (nothing yet).

```
    if request.method == 'POST':
```

- Checks if the user submitted the form.
- If yes, the method will be **POST**.
- If not (page just opened), the method is **GET**, and this block won't run.

```
    year = int(request.form['year'])
    month = int(request.form['month'])
```

- Reads the data the user entered in the HTML form:
  - `request.form['year']` → gets the “year” input from the form.
  - `request.form['month']` → gets the “month” input.
- Both are strings by default, so they’re converted to integers using `int()`.

```
cal_output = calendar.month(year, month)
```

- Uses Python’s built-in **`calendar.month()`** function.
- It generates a **multi-line text string** showing the full calendar for that month and year.

```
October 2025
```

```
Mo Tu We Th Fr Sa Su
1  2  3  4  5
6  7  8  9 10 11 12
...
```

```
return render_template('index.html', cal_output=cal_output, year=year, month=month)
```

- Renders the HTML file named **`index.html`** (from the `templates` folder).
- Also sends the variables (`cal_output`, `year`, `month`) to that HTML file, so you can display them dynamically using Jinja2 syntax like:

```
{{ cal_output }}
```

```
if name == 'main':
```

```
app.run(debug=True)
```

- This means:
  - Run the app only if this file is being executed directly (not imported elsewhere).
  - `app.run()` starts the Flask web server.
  - `debug=True` means:
    - Flask will automatically reload when you change the code.
    - It will also show error messages in the browser if something goes wrong.

Summary:

Line	Function	Meaning
`from flask import ...`	Import	Load necessary Flask tools
`import calendar`	Import	Load calendar library
`app = Flask(__name__)`	Setup	Create the Flask app
`@app.route('/', methods=['GET','POST'])`	Routing	Define the homepage
`if request.method == 'POST':`	Logic	Check if user submitted the form
`request.form[...]`	Data	Read inputs from form
`calendar.month()`	Function	Generate text calendar

`render_template()`	View	Send data to HTML page

`app.run(debug=True)`	Run	Start local web server

index.html(template folder)

`index.html` template file, which is responsible for the **frontend**

```
<!DOCTYPE html>
<html lang="en">
```

- Declares that this file is an **HTML5** document.
- `lang="en"` specifies the page's language as English

```
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>English Calendar</title>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
```

- `<meta charset="UTF-8">` : ensures the webpage supports all characters (e.g., letters, symbols, emoji 😄).
- `<meta name="viewport"...>` : makes the page **responsive**, so it looks good on phones and computers.
- `<title>` : sets the browser tab title to "English Calendar ."
- `<link rel="stylesheet"...>` : loads your CSS styling file from Flask's **static** folder.
  - `{{ url_for('static', filename='style.css') }}` is **Jinja2 syntax** : Flask replaces it with the actual URL of `style.css` .

```
<body>
<div class="container">
```

- `<body>` contains everything that will be visible on the page.
- `<div class="container">` wraps your whole app's content inside a styled box (centered, with background etc , defined in `style.css`).

```
<h1>English<span class="highlight">Calendar</span></h1>
```

- Displays the main heading (title) of your page.
- `<span class="highlight">` applies special color (like yellow) to the word "Calendar."
- The rest "English" uses normal text style.
- Both are styled using your CSS file.

```
<form method="POST">
```

- Creates a **form** where users can input data (year and month).
- `method="POST"` tells the browser to send the form data back to the Flask server via **POST request**, not visible in the URL (more secure).

```
<div class="input-group">
<label for="year">Enter a Year (2000-3000):</label>
<input type="number" name="year" id="year" min="2000" max="3000 place
holder="e.g. 2025" required>
</div>
```

- Each `<div class="input-group">` groups a label and input together neatly.

- `<label for="year">` — when clicked, focuses on the input box with `id="year"`.
- `<input type="number">` — accepts only numbers.
- `name="year"` — very important! This name is what Flask reads using `request.form['year']`.
- `placeholder` shows a light gray hint inside the box.
- `required` means the user can't submit without entering something.
- `min` and `max` restrict acceptable values

```
<div class="input-group">
<label for="month">Enter Month (1-12):</label>
<input type="number" name="month" id="month" min="1" max="12" placehol
der="e.g. 10" required>
</div>
```

- `name="month"` matches `request.form['month']` in `app.py`.
- Limits input to 1–12 for valid months.

```
<button type="submit" class="btn">Show Calendar</button>
```

- A **submit button** that sends the form data (year + month) to the backend when clicked.
- `class="btn"` allows CSS styling (color, padding, hover effects, etc.).

```
{% if cal_output %}
```

- This is a **Jinja2 template condition** — part of Flask's HTML logic system.
- It means:

Only show the following section if there's a `cal_output` value passed from Python.

- The variable `cal_output` comes from `app.py`:

```
return render_template('index.html', cal_output=cal_output, ...)
```

```
<div class="output-section">
<h2>Output:</h2>
<div class="code-box">
<pre>{{ cal_output }}</pre>
</div>
</div>
```

- isplays the result area **only after the user submits the form**.
- `<pre>` preserves whitespace and formatting (so the calendar aligns correctly like in terminal).
- `{{ cal_output }}` another Jinja2 placeholder replaced by the actual text calendar generated by Python's `calendar.month()`.
- `class="code-box"` applies the dark background and monospace font style (defined in CSS).

```
{% endif %}
```

- Closes the Jinja2 condition.
- If no calendar has been generated yet (first page load), this section is skipped.

```
  </div>
</body>
</html>
```

Summary

Section	Purpose
`<head>`	Page metadata + load CSS
`<form>`	Get user input (year + month)
`{% if cal_output %}`	Show calendar only after form submission
`<pre>{{ cal_output }}</pre>`	Display formatted calendar
`{{ url_for('static', filename='style.css') }}`	Load your external CSS file