Answer the following

1. What is CSRF protection in Django, and how does it work?

   CSRF (Cross-Site Request Forgery) is a type of attack where a malicious website tricks a user's browser into performing actions on a different website where the user is authenticated.

   CSRF Protection in Django:

   Django provides built-in CSRF protection to defend against such attacks. Here's how it works:

- **CSRF Token**: Django generates a unique token for each session. This token must be included in all POST forms submitted by the user.
- **Middleware**: The CsrfViewMiddleware checks for the presence and validity of the CSRF token in POST requests.
- **Template Tag**: In Django templates, you include the token using {% csrf_token %} inside your HTML forms.
- **Validation**: When a POST request is made, Django checks the submitted token against the one stored in the user's session or cookie. If they don’t match, the request is rejected.

2. What are Django cookies, and how do they differ from sessions?

   Cookies in Django:

   Cookies are small pieces of data stored in the user's browser. In Django, you can set, read, and delete cookies using the request and response objects.

- Set a cookie: response.set_cookie('key', 'value')
- Get a cookie: request.COOKIES.get('key')
- Cookies are stored client-side.

   Sessions in Django:

   Sessions allow you to store data on the server side, with only a session ID stored in the user's cookie.

- Django uses a session cookie (usually sessionid) to identify the user’s session.
- Session data is stored on the server (e.g., in the database, cache, or file system).
- Accessed via request.session.

Key Differences:

Feature	**Cookies**	**Sessions**
Storage	Client-side (browser)	Server-side
Security	Less secure	More secure
Data Size	Limited (generally ~4KB)	Can store more data
Persistence	Until expiry or deletion	Until expiry or logout
Use Case	Storing non-sensitive, simple data	Storing user data like login status