Django

Django Library Modeling:

1. Create a new Django project named "LibraryManagementSystem" using the `django-admin startproject` command.
2. Design the following models for your library:
   a. Book Model: Include fields such as title, author, publication date, ISBN, and availability status.
   b. Author Model: Create a model for authors with fields like name, biography, and any other relevant information.
   c. Member Model: Implement a model to represent library members with fields like name, email, contact number, etc.
3. Establish the necessary relationships between models:
   a. The Book model should have a ManyToManyField to the Author model, as a book can have multiple authors.
   b. The Member model can have a ManyToManyField to the Book model, representing the books borrowed by a member. Additionally, include fields like "borrowed_date" and "return_date" to keep track of borrowing information.
4. Register your models in the Django admin panel for easy management. Customize the admin views if necessary.
5. Run the project and access the admin panel to ensure that the models are correctly registered and functioning as expected.
6. Create a few instances of your models using the Django admin panel to test the relationships and functionalities. For example, assign authors to books, borrow and return books for members, etc.

Django

Django Calculator:

1. Set up the Django project:
    - Create a new Django project named "Calculator" using the `django-admin startproject` command.
    - Create a new Django app named "calculator_app" using the `python manage.py startapp` command.
    - Add the "calculator_app" to the project's list of installed apps in the project's settings.
2. Define URL patterns:
    - Open the project's `urls.py` file and define the URL patterns.
    - Create a URL pattern for the home page ("/"), which will be the main calculator view.
    - Create URL patterns for each arithmetic operation, such as addition, subtraction, multiplication, and division. Each pattern should include appropriate placeholders for input operands.
3. Create views:
    - Inside the "calculator_app" directory, open the `views.py` file.
    - Import the necessary modules and define the view functions for each URL pattern created in step 2.
    - Each view function should handle the corresponding arithmetic operation. Perform the calculation and pass the result to the template.
4. Design templates:
    - Create a "templates" directory inside the "calculator_app" directory.
    - Within the "templates" directory, create an HTML template named "calculator.html" for the calculator's home page. This template should include a form with input fields for operands and buttons for each arithmetic operation.
    - Create separate templates for each arithmetic operation result. Each template should display the operands, the selected operation, and the result.
5. Connect views and templates:
    - Modify the view functions to render the appropriate templates with the calculated results.
    - For the home page view, render the "calculator.html" template containing the form for inputting operands.
    - For the arithmetic operation views, render the appropriate templates that display the operands, operation, and result.

Guidelines:

- The URL pattern for the home page should be mapped to the main calculator view.
- Each arithmetic operation should have its own URL pattern and corresponding view function.

Django

- The view functions should extract the input operands from the request, perform the arithmetic operation, and pass the result to the corresponding template.
- The home page template should include a form with input fields for operands and buttons for each arithmetic operation.
- Each arithmetic operation result template should display the operands, the selected operation, and the result.

Enhancements (optional):

- Provide additional functionalities, such as support for decimal numbers or advanced operations like square root or exponentiation.
- Add client-side validation to the input fields to ensure that valid numbers are entered.
- Improve the user interface with CSS styling or JavaScript/jQuery interactivity.

Django

Django Text to HTML Converter:

1. Set up the Django project:
    - Create a new Django project named "TextToHTMLConverter" using the `django-admin startproject` command.
    - Create a new Django app named "converter_app" using the `python manage.py startapp` command.
    - Add the "converter_app" to the project's list of installed apps in the project's settings.
2. Define URL patterns:
    - Open the project's `urls.py` file and define the URL patterns.
    - Create a URL pattern for the home page ("/"), which will be the main converter view.
3. Create views:
    - Inside the "converter_app" directory, open the `views.py` file.
    - Import the necessary modules and define the view function for the home page.
    - The view function should handle the conversion of plain text to HTML.
4. Design templates:
    - Create a "templates" directory inside the "converter_app" directory.
    - Within the "templates" directory, create an HTML template named "converter.html" for the converter's home page.
    - The "converter.html" template should include a form with a textarea for the user to input the plain text.
    - Create a separate template named "result.html" to display the converted HTML result.
5. Connect views and templates:
    - Modify the view function to render the "converter.html" template for the home page.
    - Implement the conversion logic in the view function to transform the plain text input into HTML format.
    - Pass the converted HTML result to the "result.html" template for display.
6. Test the application:
    - Start the Django development server using the `python manage.py runserver` command.
    - Open a web browser and navigate to the local server's address.
    - Verify that the text-to-HTML converter web application is displayed correctly.
    - Test the converter by entering different plain text inputs and observe the generated HTML output.
7. Enhancements (optional):
    - Provide additional functionalities, such as support for basic text formatting (e.g., bold, italics, headings).

Django

- Implement client-side validation to ensure that the input textarea is not empty before conversion.
- Improve the user interface with CSS styling or JavaScript/jQuery interactivity.