Django crud code explanation in hindi :-

Record display code:-

```
def record_list(request):
    records = Record.objects.all() # डेटाबेस से सारे Records निकाल कर 'records' वेरिएबल में रख दिए।
    return render(request, 'records/record_list.html', {'records': records}) # इस डेटा को HTML पेज पर भेज दिया।
```

Record Create code :-

```
def record_create(request):
    if request.method == 'POST': # अगर यूजर ने 'Save' बटन दबाया है (डेटा भेजा है)।
        form = RecordForm(request.POST) # भेजे गए डेटा से फॉर्म भर दिया।
        if form.is_valid(): # चेक किया कि डेटा सही है या नहीं (जैसे ईमेल सही है या नहीं)।
            form.save() # डेटाबेस में नया रिकॉर्ड सेव कर दिया।
            return redirect('record_list') # वापस लिस्ट वाले पेज पर भेज दिया।
    else:
        form = RecordForm() # अगर पहली बार पेज खोला है, तो खाली फॉर्म दिखाया।
    return render(request, 'records/record_form.html', {'form': form})
```

यह रहा `record_create` फंक्शन की हर लाइन का गहराई से मतलब (Hindi Explanation):-

python
```
def record_create(request):
```

- **Meaning:** यह एक Python फंक्शन है जिसे हम 'View' कहते हैं। `request` ऑब्जेक्ट इसमें ब्राउज़र से आने वाली सारी जानकारी (जैसे URL, User डेटा) लेकर आता है।

python
```
    if request.method == 'POST':
```
Use code with caution.

- **Meaning:** यहाँ हम चेक कर रहे हैं कि यूजर ने फॉर्म भरकर "Submit" बटन दबाया है या नहीं। जब भी हम डेटा सर्वर पर भेजते हैं, तो वह `POST` मेथड के जरिए आता है।

```python
        form = RecordForm(request.POST)
```

- **Meaning:** ब्राउज़र से जो डेटा (`request.POST`) आया है, उसे हम `RecordForm` के अंदर डाल रहे हैं। यह लाइन उस डेटा को फॉर्म के "Fields" (जैसे Name, Age) के साथ जोड़ (Bind) देती है।

```python
        if form.is_valid():
```

- **Meaning:** यह एक सुरक्षा चेक है। यह देखता है कि क्या यूजर ने सारा डेटा सही भरा है? (जैसे: क्या ईमेल की जगह सही ईमेल लिखा है? क्या कोई जरूरी खाना खाली तो नहीं छोड़ दिया?)

```python
            form.save()
```

- **Meaning:** अगर डेटा सही (`valid`) है, तो यह लाइन उसे डेटाबेस में जाकर एक नई "Row" के रूप में सेव कर देती है।

```python
            return redirect('record_list')
```

.

- **Meaning:** सेव करने के बाद, हम यूजर को वापस 'record_list' वाले पेज (जहाँ सब रिकॉर्ड दिखते हैं) पर भेज देते हैं ताकि वह अपना नया रिकॉर्ड देख सके।

```python
    else:
```

- **Meaning:** यह `else` तब चलता है जब `request.method` 'POST' नहीं होता (यानी यूजर ने पहली बार पेज खोला है, फॉर्म भरा नहीं है)। इसे 'GET' रिक्वेस्ट कहते हैं।

```python
        form = RecordForm()
```

- **Meaning:** यहाँ हम एक बिल्कुल खाली फॉर्म (`Empty Form`) तैयार करते हैं ताकि यूजर के सामने एक साफ़-सुथरा फॉर्म खुले।

```python
    return render(request, 'records/record_form.html', {'form': form})
```

- **Meaning:** आखिरी में, हम Django को बताते हैं कि `record_form.html` वाला पेज दिखाओ और उसमें हमारा `form` वेरिएबल भेज दो ताकि वह HTML में दिखाई दे।

संक्षेप में (Summary):

1. अगर यूजर ने डेटा **भेजा** है (`POST`): चेक करो, सेव करो और दूसरे पेज पर भेज दो।
2. अगर यूजर सिर्फ पेज **देख रहा है** (`GET`): उसे खाली फॉर्म दिखाओ।

क्या आप समझना चाहेंगे कि `is_valid()` फेल होने पर

Record update code :-

```
def record_update(request, pk):
    # 'pk' के जरिए डेटाबेस से उस खास रिकॉर्ड को निकाला जिसे बदलना है।
    record = get_object_or_404(Record, pk=pk)

    if request.method == 'POST':
        # 'instance=record' का मतलब है कि पुराने डेटा के ऊपर नया डेटा चढ़ाओ (Update करो)।
        form = RecordForm(request.POST, instance=record)
        if form.is_valid():
            form.save() # बदलावों को सेव कर दिया।
            return redirect('record_list')
    else:
        # फॉर्म में पहले से ही पुराना डेटा भरकर दिखाया ताकि यूजर उसे बदल सके।
        form = RecordForm(instance=record)
    return render(request, 'records/record_form.html', {'form': form})
```

यह रहा `record_update` फंक्शन की हर लाइन का एकदम आसान भाषा में मतलब:

```python
def record_update(request, pk):
```

- **Meaning:** यह अपडेट करने वाला फंक्शन है। इसमें `pk` (Primary Key) आ रहा है, जो बताता है कि **किस ID** वाले रिकॉर्ड को सुधारना है (जैसे ID नंबर 5)।

```python
    record = get_object_or_404(Record, pk=pk)
```

- **Meaning:** यह लाइन डेटाबेस में जाकर उस खास रिकॉर्ड को ढूँढती है।
  - अगर रिकॉर्ड मिल गया, तो उसे `record` वेरिएबल में रख देती है।
  - अगर नहीं मिला (जैसे किसी ने गलत ID डाल दी), तो यह एरर देने के बजाय "404 Not Found" का साफ पेज दिखा देती है।

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

- **Meaning:** यह चेक करता है कि क्या यूजर ने सुधार (Edit) करने के बाद "Save" या "Update" बटन दबाया है?

```python
        form = RecordForm(request.POST, instance=record)
```

- **Meaning:** यहाँ सबसे जरूरी काम हो रहा है।
  - `request.POST`: यूजर ने जो **नया (New)** डेटा टाइप किया है।
  - `instance=record`: वह **पुराना (Old)** डेटा जो हमने ऊपर डेटाबेस से निकाला था।
  - **फायदा:** इन दोनों को साथ रखने से Django समझ जाता है कि उसे नया रिकॉर्ड नहीं बनाना, बल्कि पुराने वाले को ही अपडेट करना है।

```python
        if form.is_valid():
```

- **Meaning:** यह चेक करता है कि जो बदलाव (Changes) किए गए हैं, वे सही हैं या नहीं।

```python
            form.save()
```

- **Meaning:** यह लाइन डेटाबेस में जाकर पुराने डेटा को नए डेटा से बदल (Overwrite) देती है।

```python
            return redirect('record_list')
```

- **Meaning:** अपडेट सफल होने के बाद यूजर को वापस लिस्ट वाले पेज पर भेज दिया जाता है।

```python
    else:
```

- **Meaning:** यह तब चलता है जब यूजर ने सिर्फ "Edit" लिंक पर क्लिक किया है (अभी फॉर्म भरा नहीं है, बस पेज खोला है)।

```python
        form = RecordForm(instance=record)
```

- **Meaning:** यहाँ `instance=record` का जादू देखिए। यह फॉर्म के खानों (Inputs) में अपने आप **पुराना डेटा भरकर** दिखाता है, ताकि यूजर को पता चले कि वह क्या बदल रहा है।

```python
    return render(request, 'records/record_form.html', {'form': form})
```

- **Meaning:** आखिर में यह उस भरे हुए फॉर्म को HTML पेज (`record_form.html`) पर भेज देता है।

काम की बात (Pro Tip):

`record_create` और `record_update` में सबसे बड़ा फर्क **`instance=record`** का ही है। अगर आप इसे हटा देंगे, तो अपडेट करने के बजाय हर बार एक **नया (Duplicate)** रिकॉर्ड बन जाएगा।

Record Delete Code:-

```
def record_delete(request, pk):
    # उस रिकॉर्ड को ढूँढा जिसे डिलीट करना है।
    record = get_object_or_404(Record, pk=pk)
```

```
if request.method == 'POST': # अगर यूजर ने 'Confirm Delete' बटन दबाया।
    record.delete() # डेटाबेस से उस रिकॉर्ड को हमेशा के लिए मिटा दिया।
    return redirect('record_list')

# अगर बटन नहीं दबाया, तो डिलीट करने से पहले पूछने वाला पेज (Confirm Page) दिखाया।
return render(request, 'records/record_confirm_delete.html', {'record': record})
```

यह रहा `record_delete` फंक्शन की हर लाइन का हिंदी में विस्तृत मतलब:-

```
python
def record_delete(request, pk):
```

- **Meaning:** यह रिकॉर्ड को मिटाने (Delete) वाला फंक्शन है। यहाँ भी `pk` का इस्तेमाल किया गया है ताकि Django को पता चले कि **कौन सा वाला** रिकॉर्ड डिलीट करना है।

```
python
    record = get_object_or_404(Record, pk=pk)
```

- **Meaning:** सबसे पहले हम डेटाबेस में जाकर उस खास रिकॉर्ड को ढूँढते हैं।
  - अगर रिकॉर्ड मिल जाता है, तो वह `record` वेरिएबल में सेव हो जाता है।
  - अगर वह रिकॉर्ड मौजूद नहीं है, तो Django "404 Not Found" का पेज दिखा देता है।

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

- **Meaning:** सुरक्षा (Security) के लिए Django में हम सीधे लिंक पर क्लिक करते ही डिलीट नहीं करते। हम चेक करते हैं कि क्या यूजर ने "हाँ, मुझे डिलीट करना है" (Confirm) वाला बटन दबाया है। यह बटन हमेशा एक `POST` फॉर्म के अंदर होता है।

```
python
        record.delete()
```

- **Meaning:** यह वह लाइन है जो असली काम करती है। यह डेटाबेस में जाती है और उस खास रो (Row) को हमेशा के लिए **हटा (Remove)** देती है। एक बार यह लाइन चल गई, तो डेटा वापस नहीं आ सकता।

```python
        return redirect('record_list')
```

- **Meaning:** रिकॉर्ड डिलीट करने के बाद, हम यूजर को वापस मुख्य लिस्ट वाले पेज पर भेज देते हैं।

```python
    return render(request, 'records/record_confirm_delete.html', {'record': record})
```

- **Meaning:** अगर यूजर ने अभी तक बटन नहीं दबाया है (यानी `GET` रिक्वेस्ट है), तो हम उसे एक **सावधानी वाला पेज** (`record_confirm_delete.html`) दिखाते हैं।
    - यहाँ हम `{'record': record}` भेज रहे हैं ताकि उस पेज पर लिखा आ सके: *"क्या आप वाकई 'Rahul' का रिकॉर्ड डिलीट करना चाहते हैं?"*

निष्कर्ष (Summary):

1. **खोजो:** `get_object_or_404` से रिकॉर्ड को ढूँढो।
2. **पूछो:** `render` के जरिए कंफर्मेशन पेज दिखाओ।
3. **मिटाओ:** अगर यूजर "हाँ" कहे (`POST`), तो `record.delete()` कर दो।

यह रहा आपके `urls.py` की हर लाइन का हिंदी में आसान मतलब:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.record_list, name='record_list'),
    path('create/', views.record_create, name='record_create'),
    path('update/<int:pk>/', views.record_update, name='record_update'),
```

```
    path('delete/<int:pk>/', views.record_delete, name='record_delete'),
]
```

1. Imports (जरूरी टूल्स)

```python
from django.urls import path
```

- **Meaning:** यह Django का एक इन-बिल्ट फंक्शन है जो URL (वेबसाइट का पता) को आपके Python कोड (Views) से जोड़ता है।

```python
from . import views
```

- **Meaning:** यहाँ `.` का मतलब है "इसी फोल्डर से"। यह आपकी `views.py` फ़ाइल को इम्पोर्ट करता है ताकि हम बता सकें कि कौन सा URL कौन सा फंक्शन चलाएगा।

---

2. URL Patterns (रास्ते तय करना)

यह एक लिस्ट है जहाँ हम अपनी वेबसाइट के सारे "Address" (रास्ते) लिखते हैं।

```python
path('', views.record_list, name='record_list'),
```

- `''` : खाली स्ट्रिंग का मतलब है **Home Page** (जैसे: `://www.website.com`)।
- **`views.record_list`** : जब कोई इस पते पर आएगा, तो `views.py` का `record_list` फंक्शन चलेगा।
- **`name='record_list'`** : यह इस रास्ते का एक **Nickname** (उपनाम) है। इसे हम HTML टेम्प्लेट में लिंक देने के लिए इस्तेमाल करते हैं।

```python
path('create/', views.record_create, name='record_create'),
```

- **`'create/'`** : इसका मतलब है `://www.website.com` वाला पेज।
- **`views.record_create`** : यह नया रिकॉर्ड बनाने वाला फंक्शन चलाएगा।

```python
path('update/<int:pk>/', views.record_update, name='record_update'),
```
Use code with caution.

- **`<int:pk>/`** : यह सबसे जरूरी हिस्सा है। इसे **Dynamic URL** कहते हैं।
  - `int` : इसका मतलब है कि यहाँ सिर्फ एक नंबर (Integer) आएगा।

- `pk`: यह वह ID है जो सीधे आपके `record_update(request, pk)` फंक्शन के अन्दर जाएगी।
- **उदाहरण:** अगर आप `/update/5/` पर जाते हैं, तो Django समझ जाएगा कि ID नंबर `5` को अपडेट करना है।

```python
path('delete/<int:pk>/', views.record_delete, name='record_delete'),
```

- `'delete/<int:pk>/'`: यह डिलीट करने का रास्ता है। यहाँ भी `<int:pk>` का मतलब है उस रिकॉर्ड की ID जिसे डिलीट करना है (जैसे: `/delete/12/`)।
- `views.record_delete`: यह डिलीट करने वाला फंक्शन चलाएगा।

संक्षेप में (Summary):

- `path`: रास्ता बनाने के लिए।
- `views.function`: काम (logic) करने के लिए।
- `<int:pk>`: यह बताने के लिए कि **किस** खास रिकॉर्ड (ID) पर काम करना है।
- `name`: कोड में आसानी से उस रास्ते को बुलाने के लिए।

Templates file code:-

```
<html>

<head>

<title>Record List</title>

</head>

<body>

<h2>Record List</h2>
```

```
<a href="{% url 'record_create' %}">Add New Record</a>

<br><br>

<table border="1">

<tr>

<th>Name</th>

<th>Email</th>

<th>Phone</th>

<th>City</th>

<th>Actions</th>

</tr>

{% for record in records %}

<tr>

<td>{{ record.name }}</td>

<td>{{ record.email }}</td>

<td>{{ record.phone }}</td>

<td>{{ record.city }}</td>

<td>

<a href="{% url 'record_update' record.id %}">Edit</a>

<a href="{% url 'record_delete' record.id %}">Delete</a>

</td>

</tr>

{% endfor %}

</table>

</body>

</html>
```

यह रहा आपके **record_list.html** फ़ाइल की हर लाइन का हिंदी में मतलब:

1. HTML का ढाँचा (Basic Structure)

- `<html>`: यह बताता है कि यह एक HTML डॉक्यूमेंट है।
- `<head>`: यहाँ पेज की जानकारी होती है जो स्क्रीन पर नहीं दिखती (जैसे CSS या Title)।
- `<title>Record List</title>`: यह ब्राउज़र के टैब (Tab) पर दिखने वाला नाम है।
- `<body>`: यहाँ वह सारा कंटेंट होता है जो यूजर को स्क्रीन पर दिखाई देगा।

2. हेडिंग और लिंक

- `<h2>Record List</h2>`: पेज पर बड़ी हेडिंग दिखाने के लिए।
- `<a href="{% url 'record_create' %}">Add New Record</a>`:
  - `<a>`: यह एक लिंक बनाता है।
  - `{% url 'record_create' %}`: यह Django का जादुई टैग है। यह सीधे `urls.py` में जाता है और `name='record_create'` वाले रास्ते का पता निकाल लाता है।
  - जब आप इस पर क्लिक करेंगे, तो नया रिकॉर्ड बनाने वाला पेज खुल जाएगा।

3. टेबल की शुरुआत

- `<table border="1">`: डेटा को रो (Row) और कॉलम (Column) में दिखाने के लिए टेबल बनाई, जिसका बॉर्डर '1' है।
- `<tr> ... </tr>`: इसका मतलब है **Table Row**। यह टेबल की एक लाइन है।
- `<th>Name</th> ...`: इसका मतलब है **Table Header**। ये कॉलम के नाम हैं (Name, Email, आदि) जो थोड़े बोल्ड दिखते हैं।

4. Django का Loop (असली जादू यहाँ है)

- `{% for record in records %}`:
  - `records`: यह वह लिस्ट है जो `views.py` ने भेजी थी।
  - यह लाइन कहती है: *"डेटाबेस में जितने भी रिकॉर्ड्स हैं, उन सबको एक-एक करके दिखाओ।"*

5. डेटा दिखाना (Dynamic Content)

- `<td>{{ record.name }}</td>`:
  - `<td>`: Table Data (कॉलम के अंदर की जानकारी)।
  - `{{ ... }}`: दो बार कर्ली ब्रैकेट का मतलब है कि यहाँ Python का डेटा प्रिंट करना है।
  - यह डेटाबेस से उस व्यक्ति का 'Name' निकालकर यहाँ लिख देगा।

6. Edit और Delete बटन

- `<a href="{% url 'record_update' record.id %}">Edit</a>`:
  - यह 'Edit' का लिंक है।
  - यहाँ हमने `record.id` भी भेजा है। इसका मतलब है: *"ID नंबर X वाले रिकॉर्ड को अपडेट करने वाले पेज पर ले जाओ।"*
- `<a href="{% url 'record_delete' record.id %}">Delete</a>`:
  - यह 'Delete' का लिंक है, जो उस खास ID को डिलीट करने वाले फंक्शन के पास ले जाता है।

7. क्लोजिंग टैग्स

- `{% endfor %}`: यह Django को बताता है कि यहाँ लूप (Loop) खत्म हो गया।
- `</table>`, `</body>`, `</html>`: ये बताते हैं कि टेबल, बॉडी और पूरा पेज यहाँ खत्म हो गया है।

संक्षेप में (Summary):

यह पेज डेटाबेस से **सारे रिकॉर्ड्स** को खींचता है, उन्हें एक **टेबल** में सजाता है, और हर रिकॉर्ड के सामने **Edit** और **Delete** का बटन लगा देता है।

Record_form.html file :-

```
<!DOCTYPE html>

<html>

<head>
```

```
<title>Record Form</title>
</head>
<body>
<h2>Record Form</h2>
<form method="POST">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Save</button>
</form>
<br>
<a href="{% url 'record_list' %}">Back to List</a>
</body>
</html>
```

यह आपके **record_form.html** पेज की हर लाइन का हिंदी में मतलब है। यह वही पेज है जो **Create** और **Update** दोनों के लिए इस्तेमाल होता है:

1. HTML की शुरुआत

- `<!DOCTYPE html>`: यह ब्राउज़र को बताता है कि यह फाइल **HTML5** वर्जन में लिखी गई है।
- `<html>`, `<head>`, `<title>`: ये पेज का बुनियादी ढांचा हैं। ब्राउज़र के ऊपर वाले टैब में "Record Form" लिखा दिखाई देगा।

2. Form (डेटा भरने की जगह)

- **`<form method="POST">`**:
  - `<form>`: यह एक कंटेनर है जिसमें इनपुट बॉक्स (जैसे Name, Email) रखे जाते हैं।
  - **`method="POST"`**: इसका मतलब है कि जब यूजर बटन दबाएगा, तो डेटा **सुरक्षित तरीके से** सर्वर पर भेजा जाएगा (URL में दिखाई नहीं देगा)।

3. Django का सुरक्षा चक्र (Security)

- **`{% csrf_token %}`**:
  - यह **सबसे जरूरी** लाइन है।
  - यह एक गुप्त कोड (Token) बनाता है जो हैकर्स को आपका फॉर्म सबमिट करने से रोकता है। अगर आप इसे नहीं लिखेंगे, तो Django फॉर्म को रिजेक्ट कर देगा और "Forbidden 403" एरर दिखाएगा।

4. Form को दिखाना (The Magic Variable)

- **`{{ form.as_p }}`**:
  - `form`: यह वही वेरिएबल है जो आपने `views.py` से भेजा था (`{'form': form}`).
  - **`.as_p`**: यह Django को कहता है: *"मेरे डेटाबेस के सारे फील्ड्स (Name, Email, City) को एक के बाद एक `<p>` (Paragraph) टैग में सजाकर दिखा दो।"*
  - आपको मैन्युअल रूप से `<input>` टैग लिखने की जरूरत नहीं पड़ती।

5. Submit बटन

- **`<button type="submit">Save</button>`**:
  - `type="submit"`: यह ब्राउज़र को बताता है कि इस बटन को दबाते ही ऊपर वाले `form` का सारा डेटा सर्वर (Views) पर भेज दिया जाए।
  - `Save`: बटन पर लिखा हुआ टेक्स्ट।

6. वापस जाने का लिंक

- **`<a href="{% url 'record_list' %}">Back to List</a>`**:
  - अगर यूजर को कुछ भी सेव नहीं करना है, तो वह इस लिंक पर क्लिक करके वापस मुख्य सूची (List Page) पर जा सकता है।

एक खास बात (Pro Tip):

जब आप **Create** पेज खोलते हैं, तो `{{ form.as_p }}` आपको **खाली** डिब्बे दिखाता है। लेकिन जब आप **Update** पेज खोलते हैं, तो यही लाइन पुराने डेटा के साथ **भरे हुए** डिब्बे दिखाती है क्योंकि आपने व्यू में `instance=record` लिखा था।

record_confirm_delte.html file:-

```
<!DOCTYPE html>

<html>

<head>

<title>Delete Record</title>

</head>

<body>

<h2>Are you sure you want to delete "{{ record.name }}"?</h2>

<form method="POST">

{% csrf_token %}

<button type="submit">Yes, Delete</button>

</form>

<br>

<a href="{% url 'record_list' %}">Cancel</a>

</body>

</html>
```

यह आपके CRUD प्रोजेक्ट का आखिरी हिस्सा है—**record_confirm_delete.html**। यह पेज यूजर को गलती से डेटा डिलीट करने से बचाने के लिए एक "Warning" की तरह काम करता है।
यहाँ हर लाइन का हिंदी में मतलब दिया गया है:

1. HTML का ढाँचा (Header)

- `<!DOCTYPE html> <html> <head>`: यह ब्राउज़र को बताता है कि यह एक मानक (Standard) HTML5 पेज है।
- `<title>Delete Record</title>`: ब्राउज़र के टैब (Tab) पर "Delete Record" लिखा दिखाई देगा।

2. चेतावनी वाली हेडिंग (The Question)

- `<h2>Are you sure you want to delete "{{ record.name }}"?</h2>`:
  - `{{ record.name }}`: यहाँ Django उस खास रिकॉर्ड का नाम (जैसे: "Rahul" या "Amit") डेटाबेस से निकालकर दिखाएगा जिसे आप डिलीट करने वाले हैं।
  - यह यूजर को दोबारा चेक करने का मौका देता है कि वह सही चीज़ डिलीट कर रहा है या नहीं।

3. डिलीट करने का फॉर्म (The Action)

- `<form method="POST">`:
  - डेटाबेस से कुछ भी **मिटाने (Delete)** के लिए हमेशा `POST` मेथड का इस्तेमाल करना चाहिए। यह सुरक्षा के लिए जरूरी है ताकि कोई सिर्फ URL पर क्लिक करके आपका डेटा न उड़ा सके।
- `{% csrf_token %}`:
  - यह Django का सुरक्षा कोड (Security Code) है। इसके बिना Django फॉर्म को प्रोसेस नहीं करेगा। यह "Cross-Site Request Forgery" हमलों से बचाता है।
- `<button type="submit">Yes, Delete</button>`:
  - जैसे ही यूजर इस बटन पर क्लिक करेगा, `views.py` का `record_delete` फंक्शन चलेगा और डेटा हमेशा के लिए मिट जाएगा।

4. सुरक्षित वापसी (The Escape)

- `<a href="{% url 'record_list' %}">Cancel</a>`:
  - अगर यूजर का इरादा बदल जाए, तो वह 'Cancel' पर क्लिक करके वापस सुरक्षित तरीके से लिस्ट वाले पेज पर जा सकता है। इसमें कोई डेटा डिलीट नहीं होगा।

पूरा फ्लो (Full Process):-

यूज़र लिस्ट पेज पर **'Delete'** लिंक दबाता है।

Django उसे इस **Confirm Delete** पेज पर लाता है।

अगर यूज़र **'Yes, Delete'** दबाता है, तो `POST` रिक्वेस्ट जाती है और डेटा मिट जाता है।