DeleteとUpdate

前回のレッスンでは、フォームからメモを作成する方法を学びました。これは、CRUDのうちCreateにあたります。このレッスンでは、Update（編集）とDelete（削除）を学びましょう。このレッスンが終われば、CRUD全ての機能が実装できたことになります！

Delete機能を実装しよう

まずは、Delete（削除）機能から作りたいと思います。Delete機能の実装方法は、削除するメモを指定して、deleteメソッドを使うだけです。

削除機能を実装するために、urls.pyとviews.pyをこのように編集してください。削除するときは、特にページを表示する必要はないので、テンプレートファイルを新しく作る必要はありません。

~/memo/app/urls.py

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

app_name = 'app'
urlpatterns = [
    path('', views.index, name='index'),
    path('<int:memo_id>', views.detail, name='detail'),
    path('new_memo', views.new_memo, name='new_memo'),
    path('delete_memo/<int:memo_id>', views.delete_memo, name='delete_memo'),
]
```

一番最後にパスを追加しました。これによって`http://127.0.0.1:8000/delete_memo/(数字)`にアクセスされたときに、delete_memo関数が実行されます。

views.pyには、このようなdelete_memo関数を作ってください。

~/memo/app/views.py

```
def delete_memo(request, memo_id):
    memo = get_object_or_404(Memo, id=memo_id)
    memo.delete()
    return redirect('app:index')
```

特に新しいものは出てきていないので、この関数が何をやっているかは想像ができますよね。URLからmemo_id（数字）を受け取って、その数字とおなじIDを持つメモをdelete関数で削除しています。

これで、`http://127.0.0.1:8000/delete_memo/1`にアクセスしたときにid=1のメモが削除されるようになりました。

でもちょっと待ってください！URLを打ち込んだだけでメモが削除されてしまうのは本当はまずいですよね。ユーザーが意図しない時にメモが削除されてしまう恐れがありますし、URLにアクセスするということはHTTPメソッドでいうと**GET**になります。GETは情報の読み

取りをする時に使われるものなので、削除機能のようにDBに変更が加わる場合に使うべきではありません。

正しくは、**URLにアクセスした時ではなく、メモ詳細ページにある「削除」ボタンが押された時のみ**にメモが削除されるべきです。別の言い方をすれば、HTTPメソッドがGETの時は何の処理もしないように設定し、POSTメソッドのときにだけ削除機能が実行されるようにする必要があります。

このようにするには、以下のように関数の上に`@require_POST`を追加します。インポート文も忘れずに追加してください。

~/memo/app/views.py

```
from django.views.decorators.http import require_POST

@require_POST
def delete_memo(request, memo_id):
    memo = get_object_or_404(Memo, id=memo_id)
    memo.delete()
    return redirect('app:index')
```

これにより、delete_memo関数はrequest.method=GETの時は実行されません。
`http://127.0.0.1:8000/delete_memo/1`にアクセスしてもメモが削除されないのです。
delete_memo関数が実行されるには、リクエスト送信時のHTTPメソッドがPOSTである必要があります。

request.method=POSTにするためには、detail.htmlに以下のような削除ボタンを追加します。

~/memo/app/templates/app/detail.html

```
<div>
  <a href="{% url 'app:index' %}">ホームに戻る</a>
</div>

<h2>{{ memo.title }}</h2>

<div>
  {{ memo.text | linebreaks | urlize }}
</div>

<form method="post" action="{% url 'app:delete_memo' memo.pk %}">{% csrf_token %}
  <button class="btn" type="submit" onclick='return confirm("本当に削除しますか？");'>削除</button>
</form>
```

type="submit"としたボタンをフォームタグで囲むことによって、methodを指定できるようにしています。actionでは、'app:delete_memo'と指定していますので、削除ボタンが押された時は、HTTPメソッドがPOSTの状態で`http://127.0.0.1:8000/delete_memo/(数字)`にリクエストが送られることになります。POSTの時は、ちゃんとdelete_memo関数が実行されるのでメモも正常に削除されるというわけです。

`action="{% url 'app:delete_memo' memo.pk %}"`の`memo.pk`の部分では、
`path('delete_memo/<int:memo_id>', views.delete_memo, name='delete_memo')`の

`<int:memo_id>` に渡す数字を指定しています。これがないと、サーバー側はどのメモを削除してよいか判断できないので忘れないようにしましょう。

`onclick` ではボタンがクリックされたときにダイアログを出し、本当に削除して問題ないかをユーザーに確認しています。これがないと、ボタンが 1 回押されただけでメモが即削除されてしまうことになります。

ユーザーが保持するデータを削除するということは、Webサービスの運営上とてもデリケートなアクションですので、このように確認を取れるようにすると良いでしょう。

これで削除機能は完了です。ちょっと難しかったと思いますので、何度か読み直して頭の中を整理しながら進めてくださいね。いよいよ、残すは編集機能のみです！

メモをアップデート（編集）しよう

編集機能は、新規投稿機能を作った時とほぼ同じ流れで実装できます。編集用のページにフォームを用意し、そこから編集内容を投稿できるようにします。

まずは、編集画面を作成しましょう。urls.pyには編集画面用のパスを追加してください。

~/memo/app/urls.py

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

app_name = 'app'
urlpatterns = [
    path('', views.index, name='index'),
    path('<int:memo_id>', views.detail, name='detail'),
    path('new_memo', views.new_memo, name='new_memo'),
    path('delete_memo/<int:memo_id>', views.delete_memo, name='delete_memo'),
    path('edit_memo/<int:memo_id>', views.edit_memo, name='edit_memo'),
]
```

edit_memo関数はこのようになります。

~/memo/app/views.py

```
def edit_memo(request, memo_id):
    memo = get_object_or_404(Memo, id=memo_id)
    form = MemoForm
    return render(request, 'app/edit_memo.html', {'form': form, 'memo':memo })
```

edit_memo.htmlは新たに作成してください。new_memo.htmlと同じように{{ form.as_p }}の表記で、edit_memo関数で作ったフォームを表示します。

~/memo/app/templates/app/edit_memo.html

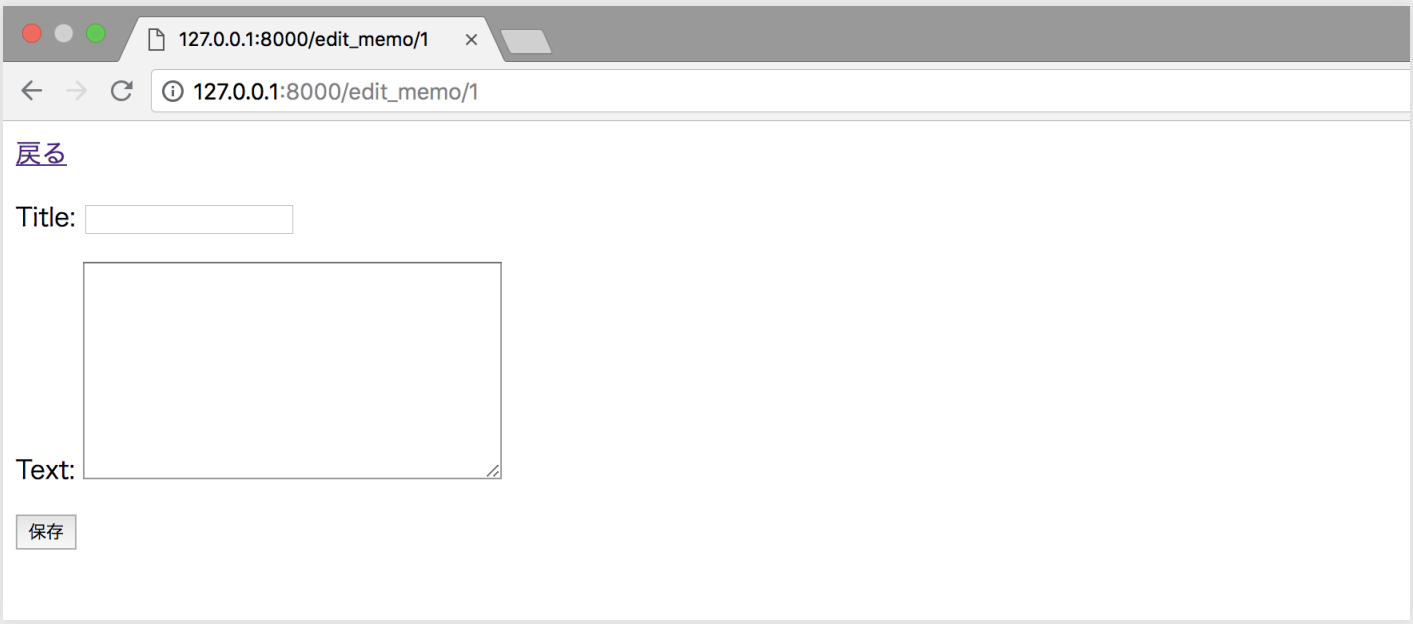
```
<div>
  <a href="{% url 'app:detail' memo.id %}">戻る</a>
</div>

<form action="{% url 'app:edit_memo' memo.id %}" method="POST">{% csrf_token %}
  {{ form.as_p }}
  <button class="btn" type="submit">保存</button>
</form>
```

detail.htmlには、編集画面へのリンクを追加しておきましょう。

~/memo/app/templates/app/detail.html

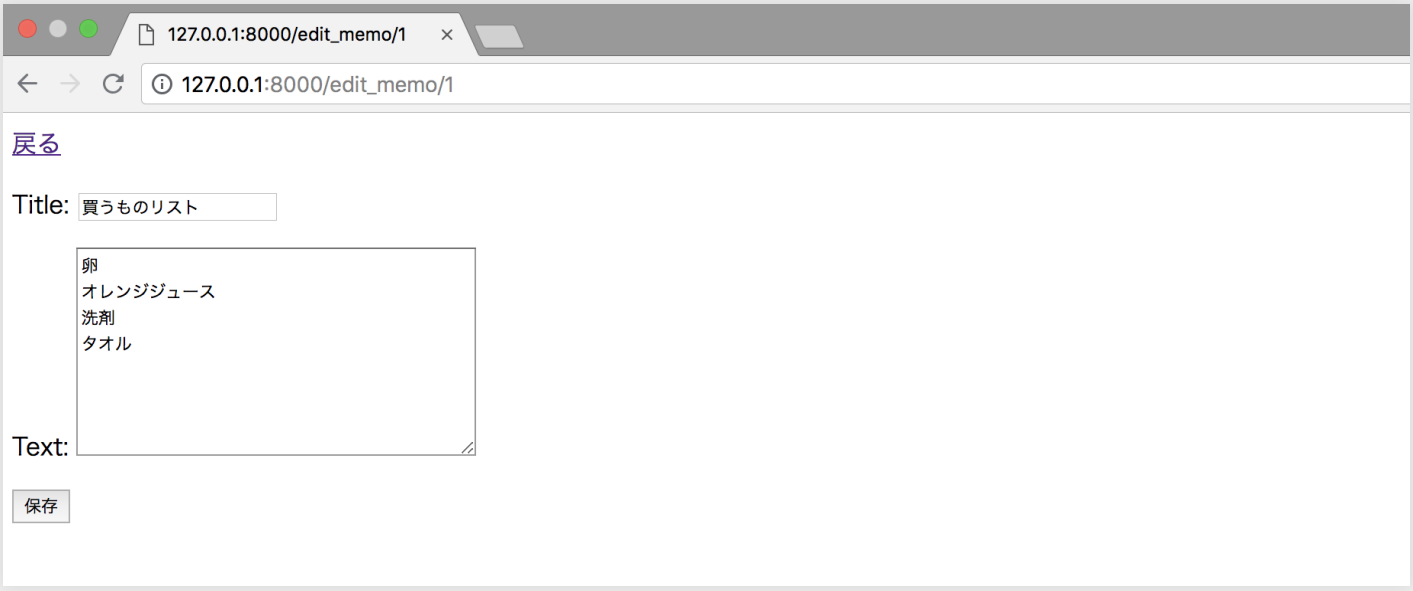
```
<div>
  <a href="{% url 'app:index' %}">ホームに戻る</a>
</div>

<h2>{{ memo.title }}</h2>

<div>
  {{ memo.text | linebreaks | urlize }}
</div>

<button><a class="btn" href="{% url 'app:edit_memo' memo.pk %}">編集</a></button>

<form method="post" action="{% url 'app:delete_memo' memo.pk %}">{% csrf_token %}
  <button class="btn" type="submit" onclick='return confirm("本当に削除しますか？");'>削除</button>
</form>
```

これで、編集画面ができました。やっていることとしては、新規投稿機能とほぼ同じことですので、それと似たようなページができます。



しかし、編集画面というのは既に保存されてあるメモの内容を修正するページですので、フォームが空欄のまま表示されてしまうのはよくありません。フォームの入力欄には、保存されてある文章が表示されるようにしておき、それを修正できるようにすべきです。

↓本来のイメージ図



上図のように、保存されている内容がフォームに入っている状態で表示するためには、views.pyを以下のように編集します。

~/memo/app/views.py

```
def edit_memo(request, memo_id):
    memo = get_object_or_404(Memo, id=memo_id)
    form = MemoForm(instance=memo)
    return render(request, 'app/edit_memo.html', {'form': form, 'memo':memo })
```

MemoFormに`instance`という引数を与え、右辺では１行上で取得しているMemoインスタンスを指定しています。これにより、指定したインスタンスに対応したフォームが作成されます。変更を保存したら、編集画面をリロードしてみましょう。

うまく入力欄に値が表示されましたか？このようにModelFormは、`ModelForm(instance=インスタンス)`とすることで、あらかじめインスタンスの値を持ったフォームを表示させることができるのです。

さあ、編集画面の表示はこれで完成です。あとは、新規投稿画面と同じように、保存ボタンが押された時（HTTPメソッドがPOSTのとき）の処理を加えましょう。

edit_memo関数をこのように編集してください。

~/memo/app/views.py

```
def edit_memo(request, memo_id):
    memo = get_object_or_404(Memo, id=memo_id)
    if request.method == "POST":
        form = MemoForm(request.POST, instance=memo)
        if form.is_valid():
            form.save()
            return redirect('app:index')
    else:
        form = MemoForm(instance=memo)
    return render(request, 'app/edit_memo.html', {'form': form, 'memo':memo })
```

new_memo関数では、`form = MemoForm(request.POST)`として、受け取った情報を元に新しいMemoインスタンスを生成していました。ここでは`form = MemoForm(request.POST, instance=memo)`のようにすることで、受け取った情報を元に既存のMemoインスタンスを上書きするようにしています。

あとは、new_memo関数と同じようにフィールドの値が正常であればDBに保存してリダイレクトさせるという流れです。

お疲れ様でした！これでメインとなる機能は全て実装することができました！最後に、フォームの表示の仕方について、ちょっとだけ別のやり方を紹介しますね。

フォームを詳細に表示する

これまで、テンプレートファイルでフォームを表示するときは、`{{ form.as_p }}`のような書き方で表示させてきましたよね。この表記だけで、フォームが表示できてしまうのはとても便利なのですがフォームのラベルや入力欄にCSSを効かせてデザインすることができないので、ちょっと不便な点もあります。それに、ラベルのところは自動でフィールド名が表示されるので、TitleやTextのように英語で表示されてしまっています。

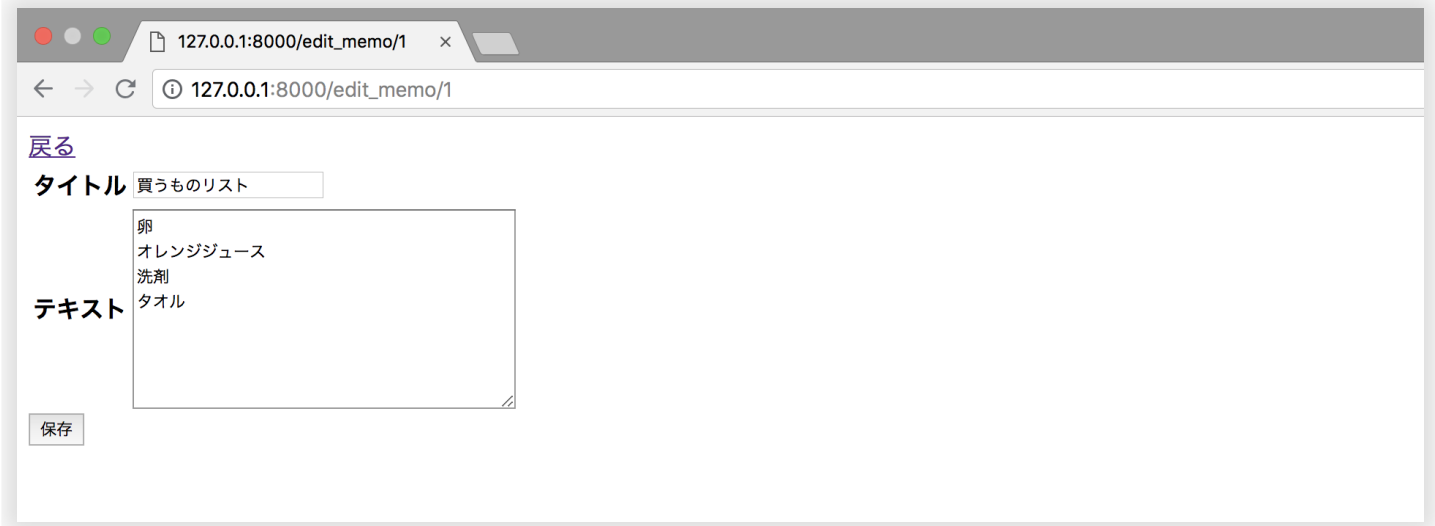
実は、フォームは以下のような書き方でも表現できますので、formタグの中身を書き換えて試してみましょう。

~/memo/app/templates/app/edit_memo.html

```
<div>
<a href="{% url 'app:detail' memo.id %}">戻る</a>
</div>

<form action="{% url 'app:edit_memo' memo.id %}" method="POST">{% csrf_token %}
  <table>
    <tr>
      <th>タイトル</th>
      <td>{{ form.title }}</td>
    </tr>
    <tr>
      <th>テキスト</th>
      <td>{{ form.text }}</td>
    </tr>
  </table>
  <button class="btn" type="submit">保存</button>
</form>
```

`{{ form.text }}`のように、`{{ form.フィールド名 }}`とすれば、フィールド名に対応した入力欄が表示されます。

これで、ラベル名は自由に付けられるようになりましたね。今回は「タイトル」のように単純にカタカナで表記しています。また、より細かいタグの単位でCSSを適用できるようになりました。



ちなみにですが、`{{ form.title.label }}`で、フィールド名がラベルとして使えます。この例の場合は、`Title`と表記されますので、フィールド名をそのままラベルに使いたい場合は、この書き方でも良いかもしれませんね。

うまく表示できたら、new_memo.htmlの{{ form.as_p }}も同じ書き方で書き換えてみましょう！hrefやactionの指定先は編集画面と新規投稿画面で異なりますので注意してくださいね。

なかなかボリュームのあるレッスンでしたが、うまく理解できたでしょうか。このレッスンを習得できればCRUDは完璧です！

いよいよ次は最後のレッスンです。より効率よく開発できる手段を学びましょう！