

Djangoテンプレート言語の基本構文

変数の表示

テンプレート内で変数を表示するには、二重中括弧 `{{ }}` を使用します。

```
{{ 変数名 }}
```

属性の表示

変数の属性を表示するには、ドット `.` を使用します。

```
{{ 変数名.属性名 }}
```

フィルターの使用

フィルターを使用して変数の値を変換できます。フィルターはパイプ `|` で指定します。

```
{{ 変数名 | フィルタ名 }}
```

forループ

リストの各要素に対してループを実行するには、`{% for %}` タグを使用します。

```
{% for 変数名 in リスト %}  
    <!-- ループ内の処理 -->  
{% endfor %}
```

if文

条件分岐を行うには、`{% if %}` タグを使用します。

```
{% if 変数 %}  
    <!-- 条件が真の場合の処理 -->  
{% else %}  
    <!-- 条件が偽の場合の処理 -->  
{% endif %}
```

URLの逆引き

URLの逆引きを行うには、{% url %} タグを使用します。

```
{% url '名前' 変数,変数... %}
```

静的ファイルの読み込み

静的ファイルをテンプレートに含めるには、{% load static %} タグを使用します。

```
{% load static %}
```

静的ファイルのパス

静的ファイルのパスを取得するには、{% static %} タグを使用します。

```
{% static 'パス' %}
```

CSRFトークン

フォームにCSRFトークンを含めるには、{% csrf_token %} タグを使用します。

```
{% csrf_token %}
```

テンプレートエンジン

- Djangoでは、HTML等の雛形（テンプレート）を予め作っておくことで、出力時に動的に変数などを代入する仕組みがある
- テンプレートは「Djangoテンプレート言語」で記述する
 - 「雛形」なので、最終的に出力したいHTML等の中に、決まったルールで、動的な部分を記述する

テンプレート言語：変数とフィルタ

- 変数
 - `{{ 変数名 }}`
変数の値で置き換える
 - `{{ 変数名.属性名 }}`
変数の属性の値で置き換える
 - 辞書であれば、属性名をキーとした辞書の値
 - オブジェクトであれば、オブジェクトの属性
 - 例)
 - `{{ article.title }}`
テンプレートエンジンに渡されている `article` の `title` 属性に置き換わる
- フィルタ
 - `{{ 変数名 | フィルタ名 }}`
変数の値に、以下のようなフィルタ処理を行った結果で置き換える
 - `lower` : 小文字にする
 - `linebreaksbr` : 改行を `
` タグに置き換える
 - etc...
 - 例)
 - `{{ article.text | linebreaksbr }}`
テンプレートエンジンに渡されている `article` の `text` 属性の改行を `
` に置き換えた結果に置き換わる

テンプレート言語：繰り返しと条件分岐

- 繰り返し
 - `{% for 変数名 in リスト %}`
何らかの処理
`{% endfor %}`
リストの要素に順に処理を適用する
 - 例)
 - `{% for article in articles %}`
 `<p> {{ article.title }} </p>`
`{% endfor %}`
テンプレートエンジンに渡されている `articles` の各要素の `title` 属性を順に出力する
- 条件分岐
 - `{% if 変数 %}`
処理A
`{% else %}`
処理B
`{% endif %}`
変数の値が存在し、空のリストでもFalseでもなければAを適用し、それ以外の場合はBを適用する

テンプレート言語を試してみましょう

- テンプレート言語テスト用に、以下の手順で新しいページを追加する
 1. URLディスパッチャの追加
→ `blog/urls.py`
 2. コントローラにおけるリクエストの処理の追加
→ `blog/views.py`
 3. テンプレートの追加
→ `blog/templates/blog/hello.html`

手順①URLディスパッチャの追加 : `blog/urls.py`

- URLディスパッチャが、以下のURLを受け付けるように変更したい
 - `http://127.0.0.1:8000/hello` → `views.py` の `hello` 関数を呼び出す
- そのために、`urlpatterns` リストの最後に、次の内容を追加する
 - `path('hello', views.hello, name='hello')`
 - `'hello'` : マッチするパターン
 - `views.hello` : マッチした時に呼び出す関数
 - `name='hello'` : このパターンの名前

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     ...
6     path('hello', views.hello, name='hello'),
7 ]
```

手順②Controllerへの処理の追加：blog/views.py

- **blog/views.py** の最後に以下のコードを追記し、**hello** 関数を定義する

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  # Create your views here.
5  def index(request):
6      return render(request, 'blog/index.html')
7
8  def hello(request):
9      return render(request, 'blog/hello.html')
10
```

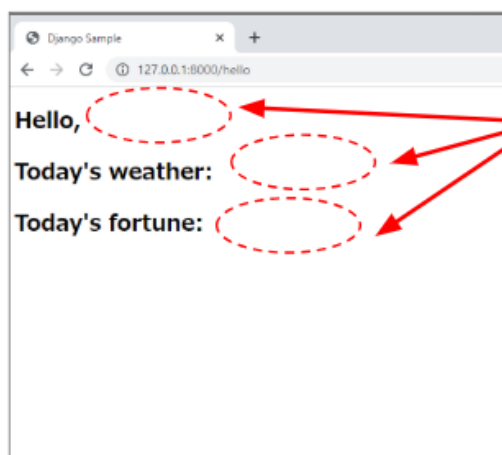
手順③テンプレートの追加：

- **blog/templates/blog** フォルダの中に以下の **hello.html** を作成する

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Django Sample</title>
6      </head>
7      <body>
8          <h2>Hello, </h2>
9          <h2>Today's weather: </h2>
10         <h2>Today's fortune: </h2>
11     </body>
12 </html>
13
```

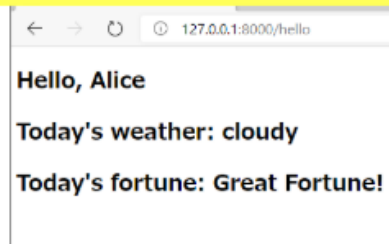
Viewの表示を確認しましょう

- `http://127.0.0.1:8000/hello`にアクセスしてみましょう
 - 以下のようなページが表示されましたか？



この部分は動的にデータを埋め込み
たい！
→ テンプレート言語を利用しよう

このようなページにしたい



テンプレートを修正

： `blog/templates/blog/hello.html`

- 以下のように動的にデータを埋め込む部分に変数を追加する

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Django Sample</title>
6    </head>
7    <body>
8      <h2>Hello, {{ name }}</h2>
9      <h2>Today's weather: {{ weather }}</h2>
10     <h2>Today's fortune: {{ fortune }}</h2>
11   </body>
12 </html>
13
```

Controller からViewへ変数の受け渡し

- `render`関数の第三引数に、辞書型のオブジェクトを指定することで、View(テンプレート)に値を渡すことができる
 - `render(request, template, context)`
 - `request` : HTTPRequestオブジェクトを渡す
 - `template` : テンプレートファイルを指定する
 - `context` : テンプレートに受け渡す値を、辞書型オブジェクトで与える

Controllerを修正：blog/views.py



- 以下のようにテンプレートに渡す値を辞書型オブジェクトとして宣言する

```

1 from django.shortcuts import render
2 from django.http import HttpResponseRedirect
3
4 # Create your views here.
5 def index(request):
6     return render(request, 'blog/index.html', {'name': 'Alice'})
7
8 def hello(request):
9     data = {
10         'name': 'Alice',
11         'weather': 'CLOUDY',
12         'fortune': 'Great Fortune!'
13     }
14     return render(request, 'blog/hello.html', data)
15

```

- テンプレートに渡す値は、KeyとValueで構成された辞書型オブジェクト
- テンプレート内で、Keyの名前で、対応するValueを参照することができる

render メソッドの第三引数として、テンプレートに渡す辞書型のオブジェクトを指定

Viewの表示を確認しましょう



- `http://127.0.0.1:8000/hello`にアクセスしてみましょう
 - 渡した値を反映したページが表示されましたか？

```

def hello(request):
    data = {
        'name': 'Alice',
        'weather': 'CLOUDY',
        'fortune': 'Great Fortune!'
    }
    return render(request, 'blog/hello.html', data)

```

Controller(blog/views.py)

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Django Sample</title>
</head>
<body>
<h2>Hello, {{ name }}</h2>
<h2>Today's weather: {{ weather }}</h2>
<h2>Today's fortune: {{ fortune }}</h2>
</body>
</html>

```

View(blog/templates/blog/hello.html)

渡された辞書型オブジェクトのKey "name"に対応するValue "Alice"に置き換わる

Hello, Alice
Today's weather: CLOUDY
Today's fortune: Great Fortune!

ブラウザに表示される画面

テンプレート言語のフィルタ適用例



- weather変数の値に「lower」フィルタを適用すると、値を小文字に変換した結果に置き換わる
 - 他にも様々なフィルタが用意されている
<https://docs.djangoproject.com/ja/3.2/ref/templates/builtins/#ref-templates-builtins-filters>

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Django Sample</title>
6 </head>
7 <body>
8 <h2>Hello, {{ name }}</h2>
9 <h2>Today's weather: {{ weather | lower }}</h2>
10 <h2>Today's fortune: {{ fortune }}</h2>
11 </body>
12 </html>
13

```

Django Sample x +
127.0.0.1:8000/hello
Hello, Alice
Today's weather: cloudy
Today's fortune: Great Fortune!



テンプレート言語のタグ適用例：繰り返し

- 以下のようにControllerを修正し、辞書型オブジェクトに要素を追加する

このようなページにしたい

- Keyが "weather_detail"
- Valueが、3個の文字列型の要素を持つリスト

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4
5
6
7
8 def hello(request):
9     data = {
10         'name': 'Alice',
11         'weather': 'CLOUDY',
12         'weather_detail': ['Temperature: 23°C', 'Humidity: 40%', 'Wind: 5m/s'],
13         'fortune': 'Great Fortune!'
14     }
15     return render(request, 'blog/hello.html', data)
16
```

気象の詳細を示す項目を順に並べたい

Hello, Alice

Today's weather: cloudy

- Temperature: 23°C
- Humidity: 40%
- Wind: 5m/s

テンプレート言語のタグ適用例：繰り返し



- テンプレート内でforタグを利用することで、Controllerから渡されたリストの要素に順に処理を適用することができる

渡された辞書型オブジェクトのKey "weather_detail" に対応するValue
["Temperature: 23°C", "Humidity: 40%", "Wind: 5m/s"] に置き換わる

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
<title>Django
</title>
<head>
</head>
<body>
<h2>Hello, {{ name | title }}</h2>
<h2>Today's weather: {{ weather | lower }}</h2>
<ul>
{% for item in weather_detail %}
<li>{{ item }}</li>
{% endfor %}
</ul>
<h2>Today's fortune: {{ fortune }}</h2>
</body>
</html>
```

Key "weather_detail" に対応するリストの要素が
順に変数itemに代入され、出力される

Hello, Alice

Today's weather: cloudy

- Temperature: 23°C
- Humidity: 40%
- Wind: 5m/s

Today's fortune: Great Fortune!



テンプレート言語のタグ適用例：条件分岐

- 以下のようにControllerを修正し、辞書型オブジェクトに要素を追加する

```

1 from django.shortcuts import render
2 from django.http import HttpResponse
3 import random
4
5 # Create your views here.
6 def index(request):
7     return render(request, 'blog/index.html')
8
9 def hello(request):
10     data = {
11         'name': 'Alice',
12         'weather': 'CLOUDY',
13         'weather_detail': ['Temperature: 23°C', 'Humidity: 40%', 'Wind: 5m/s'],
14         'isGreatFortune': True,
15         'fortune': 'Great Fortune!'
16     }
17
18     return render(request, 'blog/hello.html', data)

```


このようなページにしたい

Hello, Alice

Today's weather: cloudy

条件が True なら王冠を、False なら別の画像を出力したい

Today's fortune: Great Fortune!



Keyが "isGreatFortune"、ValueがTrue

テンプレート言語のタグ適用例：条件分岐



- テンプレート内でifとelseタグを利用することで、Controllerから渡された値によって出力するコンテンツを切り替えることができる

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Django Sample</title>
6 </head>
7 <body>
8 <h1>Hello, {{ name }}</h1>
9
10 <h2>Today's weather: {{ weather }}</h2>
11
12 <div>
13 <ul>
14 <li>Temperature: {{ weather_detail.0 }}</li>
15 </ul>
16 <h2>Today's fortune: {{ fortune }}</h2>
17 <div>
18 
19 
20 <div>
21 
22 </div>
23 </body>
24 </html>

```

Key "isGreatFortune" に対応する値が True のため、王冠画像が出力される。値が False の場合は、応援画像が出力される

王冠画像のURL

応援画像のURL

Hello, Alice

Today's weather: cloudy

Temperature: 23°C
Humidity: 40%
Wind: 5m/s

Today's fortune: Great Fortune!





テンプレート言語：その他のタグ

- URL
 - `{% url '名前' 変数, 変数... %}`
URLディスペッチャ内の、指定した名前(`name=`で指定した値)に対応するパターンに置き換える
 - 変数をパターンに代入する
 - 例)
 - `{% url 'detail' article.id %}`
: 'detail'という名前のURLパターンに置き換える
→ /数字
- その他、サンプルで利用しているタグ
 - `{% load static %}`
静的ファイルを読み込めるようにする
 - `{% static 'パス' %}`
静的ファイルのURLに置き換える
 - `{% csrf_token %}`
フォーム内で、CSRF対策のためのトークンを出力する



テンプレート言語：静的ファイルについて

- アプリケーションフォルダの`static`フォルダ以下に置いたファイルは、`'/static/'`から始まるパスでアクセスできる
 - テンプレートの1行目に、以下を記述する
 - `{% load static %}`
 - テンプレート内のタグは以下のように展開される
 - `{% static 'css/bootstrap.css' %}`
 - 展開されるURL : </static/css/bootstrap.css>
 - 参照される静的ファイル : `blog/static/css/bootstrap.css`
- 
- `static`フォルダの作成及び静的ファイルの読み込みは今後の授業で行います

CSRF対策について

- CSRF (Cross-site request forgeries) : クロスサイトリクエストフォージェリ
 - Webアプリケーションに対する攻撃の一種
 - 一般に、悪意のあるウェブサイトにおいて、画像読み込み、リンクのクリック、フォーム投稿などを行った際に、ユーザが意図しないリクエストをWebアプリケーションに対して行わせる（例えば、勝手に投稿させる等）攻撃のことを指す
- この対策として、フォームを生成する際にランダムな値を発行し、リクエストの際に、この値を含めることで、正規のリクエストであることを検証する
- Djangoでは、フォーム内に以下のタグを必ず含める
 - `{% csrf_token %}`

CSRFの例

