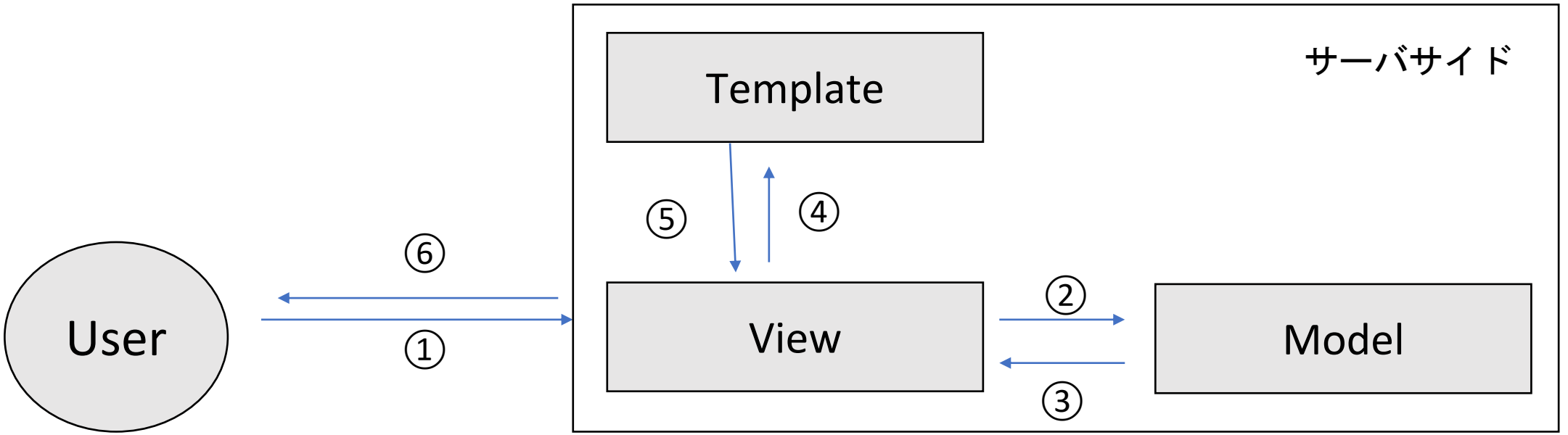
Django基本講座２
(templateの利用)

Templateとは

ユーザ（View）からの要求に応じて動的にページを作成して返す機能を持っている。
HTML、CSS、JavaScriptなどの一般的なWebサイトの生成と同時にPythonのコードを埋め込むことで、Pythonで生成した値を画面上に出力することができる



Templateの使用

アプリケーションのフォルダ配下にtemplatesというフォルダを作成して中に、htmlファイルを格納していく

```
first_app/
    templates/  ・・・フォルダ
        first_app/・・・フォルダ
            index.html ・・・htmlファイル
    views.py・・・views.pyでhtmlファイルを指定して開く
```

views.pyに以下のように記述する

```
from django.shortcuts import render

render(request, ‘first_app/index.html’) # first_appフォルダ内のindex.htmlを表示
```

Templateのディレクトリの変更

settings.pyの以下の部分にDjangoが認識するtemplatesのディレクトリを指定

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],←templateとして利用するディレクトリを記載
```

以下のようにすると、環境がWindowsからLinuxなどに変わってもプロジェクトフォルダ直下(manage.pyと同じ階層)のtemplatesディレクトリを指すようになる

```
import os

TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
‘DIRS’: [TEMPLATE_DIR, ],
```

値を渡してtemplateで使用する

render実行時に引数として、context=辞書として値を取る

```
    return render(request, 'file.html', context={‘value’: ‘HELLO’})
```

templates側で、渡された値を利用する

file.html

<h1>{{ value }}</h1> # HELLOが表示される

Django template language(DTL)とは？

DTL: デザインを容易にすることができるPythonのテンプレートエンジン。PythonをHTMLで利用するためのツール。Pythonのテンプレートエンジンは、Flaskで利用されるJinjaなどもあり、記述方法も似ています。

views.py

```
def index(request):
    my_name = "my Name" # 変数として設定
    letters = list(my_name)   # リスト型として設定
    human_dic = {'name': 'taro'} # 辞書型として設定
    return render(request, 'base.html', context={
        'my_name': my_name, 'letters': letters, 'human_dic': human_dic # 変数をtemplateに渡す
    })
```

 # 変数my_name, letters, human_dicを渡す

template側

```
<h1>{{my_name}}</h1> {# 変数をそのまま表示 #}
<h1>{{letters}}</h1> {# リスト型をそのまま表示 #}
<h1>{{human_dic.name}}</h1> {# 辞書型をそのまま表示 #}
```

DTLの様々な制御文

{% %}: if, forなどの式（制御文）を記載
{{ }}: 値をアウトプットする
{# #}: コメント文

for文

```
{% for value in mylist %}
<p>{{ value }}</p>
{% endfor %}
```

コメント文

```
{# ... #}
```

if文

```
{% if value in mylist %}
<p> something</p>
{% elif 式 %]

{% else %}
<p>Hmmm</p>
{% endif %}
```

DTLでの他のtemplateの継承

DTLでは、他のテンプレートを継承して、そのテンプレートの機能をそのまま利用することができます。継承元に、jquery, bootstrapなど様々なライブラリの読込み、ヘッダーの作成など、画面間で共通する機能を実装し、継承先で独自の内容を記述するようにします

継承元

```
<html>

{% block content %}
ここの内容は継承先が記載する
{% endblock %}

</html>
```

継承先

```
{% extend “base.html” %}

{% block content %}
ここに独自の内容を記述する
{% endblock %}
```

templateのフィルター機能

フィルターを用いると、template上で
値の変換をすることができます。

文章全体を指定
{% filter upper %} ~ {% endfilter %}

各単語をフィルタ
{{ variable | filter }}

代表的なフィルター一覧。詳細は下記
https://docs.djangoproject.com/ja/3.1/ref/templates/builtins/#built-in-filter-reference

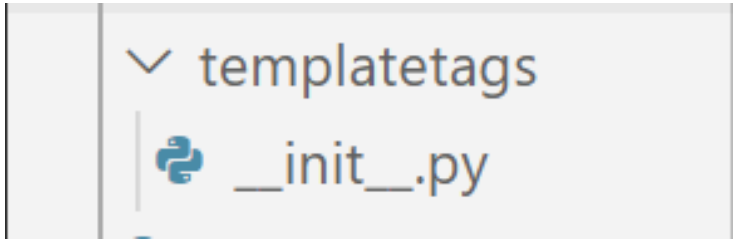
add →値を追加する	filesizeformat →MBのようなファイルサイズフォーマットに変換する	lower →小文字に変換	urlizetrunce →urlを遷移可能にし、文字数を指定以上で切り詰める
addslashes →引用符の前に¥を加えます	first →リスト中の最初の要素	random →シーケンスからランダムに要素を取り出す	urlize →文字列の中にあるurlが遷移できるようになる
capfirst →最初の文字を大文字にする	floatformat →小数点以下を丸める	safe →HTMLエスケープが必要でない場合にマークします	linebreaksbr →改行をbrにする
cut →指定した値を取り除く	join →配列を指定した文字を挟んで結ぶ	slice →リストの一部を返します	escape() →HTMLをエスケープします
default →デフォルト値を指定	last →リストの最後の要素を取り出す	truncatechars →文字を途中で切り詰めてのころを.....にします	
date →指定した日付でフォーマットします	length →リストの要素の長さを返す	upper →大文字に変換します	
dictsort →引数で指定したキーでソートします	linebreaks →改行をbrにしpタグを付与する変換する	urlencode →入力値をURLとして使えるようにエンコードします	

templateでのfilterの自作

templateで使用するfilterを自作することもできます。

【方法】

1. アプリケーションのフォルダ内に**templatetags**フォルダを作成する



2. templatetagsの中に**__init__.py**を作成

3. filterを自作するようのファイルを作成し、中に関数を記述し
   from django import template
   register = template.Library()

   @register.filter(name='自作したフィルタの名前')
   def 関数名(value):

4. templateの中で**loadして利用する**。

   {% load 作成したfilterのファイル名 %}

templateでの画面遷移

templateを用いて画面遷移を行う場合には、{%url %}を用います。

例えば、
アプリケーションフォルダ内のurls.pyには、以下の記述があるとします。

app_name = '**app**’

urlpatterns = [
    path('home', views.home, name='**home**'),
]

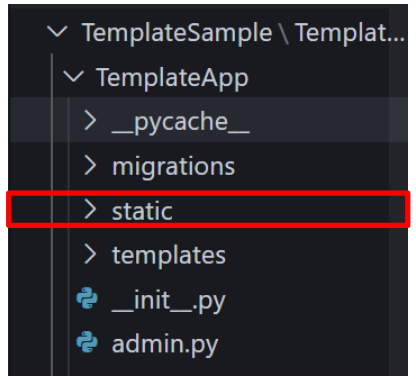
この場合、app.homeのURLにアクセスするには、

テンプレート側で以下の記述を行います。
Home

Viewに値を渡す場合
Link

templateでのstaticの利用

staticというのは、cssファイル, jsファイル, 画像などの静的なコンテンツを入れておくためのディレクトリで
Djangoでは静的なコンテンツを入れたフォルダから画像を取り出します。



まず、templateに以下のように記載します。

```
{% load static %}
<img src="{% static "my_app/example.jpg" %}" alt="My image"> # 画像の読込み
<link rel="stylesheet" type="text/css" href="{% static 'my_app/style.css' %}"> # CSSの読込み
```

デフォルトでの静的コンテンツの配置場所は、アプリケーションのstaticというフォルダです。

静的コンテンツを配置するフォルダを変更する場合には、settings.pyに以下の記述を追加します。

```
STATICFILES_DIRS = [
    'staticフォルダのパス'
]
```

または、

```
STATICFILES_DIRS = [
    ('フォルダの識別子', 'staticフォルダのパス')
]
```

インスタンスの中のプロパティを表示

templateを用いてクラス内のプロパティを取り出して画面上に表示することもできます。

例えば、
views.pyには以下のように記述します。

```
def home(request):
    instance = Class(propery1=〇〇, …)
    return render(request, ‘template.html’, context={})
```

そして、templateでは、以下のようにするとプロパティにアクセスして中の値を取り出せます。

template.html

```
{{ instance.propery1 }} {# プロパティにアクセス #}
```

問題

ちょっとした、ホームページを作成します

1. TemplateExamというプロジェクトを作成しましょう
2. TemplateAppというアプリケーションを作成しましょう
3. migrateを行って、Djangoのデフォルトのデータベースを作成しましょう
4. TemplateAppの中に以下の画面を作成して、それぞれ各URLで遷移できるようにしましょう
   4-1. http://127.0.0.1:8000/app/home
   → サークルのホーム画面です。Django大学陸上部 ホームとだけ表示されます
   4-2. http://127.0.0.1:8000/app/members
   → サークルのメンバーが表示されます。Taro,Jiro,Hanako,Yoshikoさんがいます
   4-3. http://127.0.0.1:8000/app/member/{id}
   → メンバーの詳細画面が表示されます。名前、顔写真、入部日(yyyy/mm/dd)、入部から何年何カ月経過しているかが表示されます。この画面は、メンバー一覧ページからメンバーの名前を選択すれば遷移できます。各メンバーの顔写真はUdemyに添付しています。