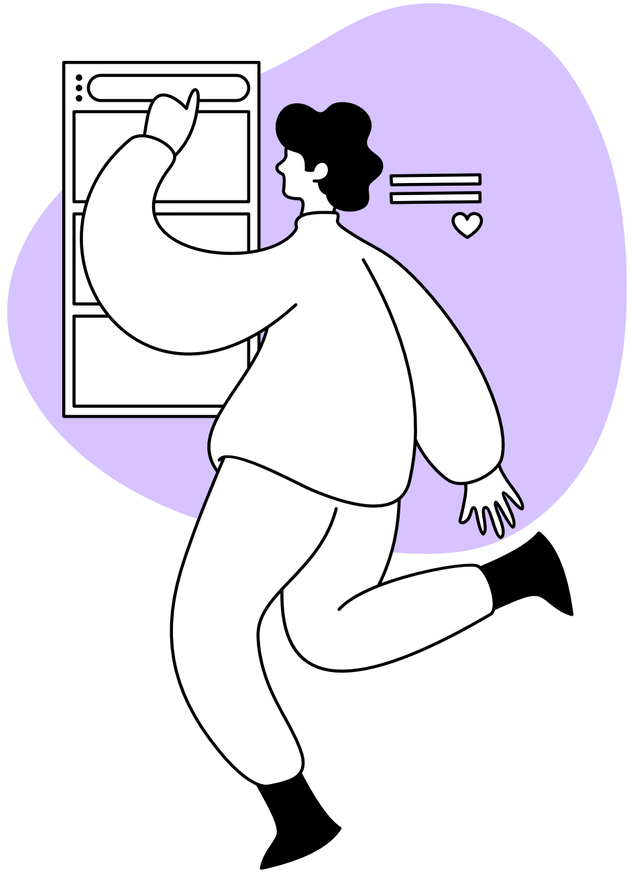
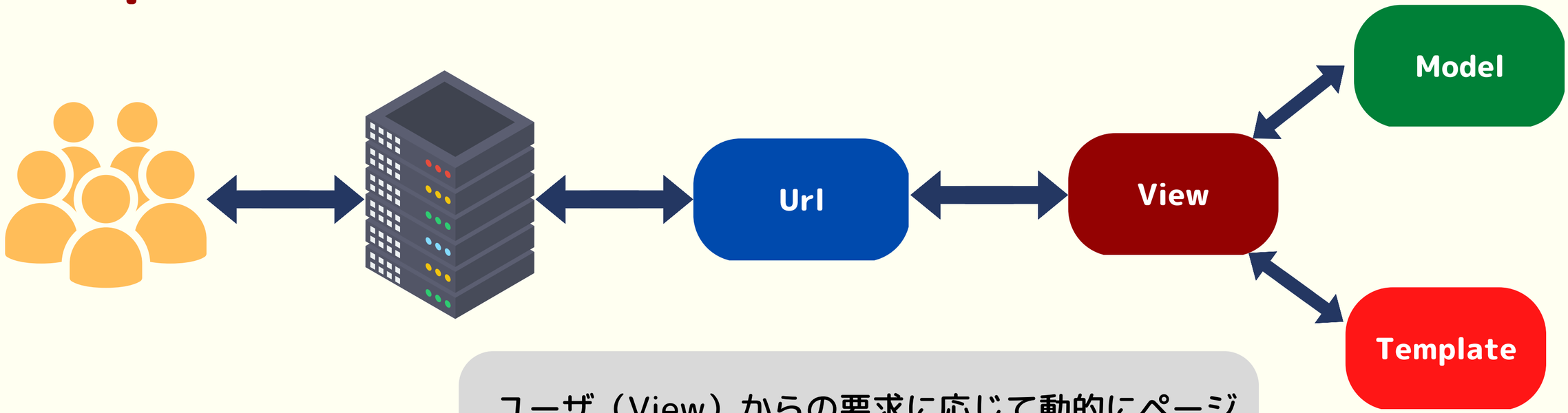
Django Template

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。HTML, CSS, JavaScriptなどの一般的な静的ファイルとPythonのコードを埋め込み、Viewで生成した値を画面上に表示する

Model: DBにアクセスし、データの挿入、更新、取得などをする
View: 入力を受けて、Modelの呼び出しや処理を行い、Templateをユーザに返す
Template: ユーザに応じて、動的にページを作成して画面表示する

Templateの使用方法

アプリケーションのフォルダ配下にtemplatesフォルダを作成して、中にhtmlファイルを格納する

フォルダ構成

```
app/
   templates/
      first_app/
         index.html
    views.py
```

views.pyにてrender関数で、テンプレートのパスを指定する(templatesの指定は不要)
URLディスパッチャで指定した、URLにアクセスすると、indexが呼び出され、first_app/index.htmlを表示する

views.pyの記述内容

```
from django.shortcuts import render

def index(request):
   render(request, 'first_app/index.html') # 表示するhtmlファイルを指定する
```

Templateの配置ディレクトリの変更

設定ファイル(settings.py)にディレクトリを指定する

```
TEMPLATES = [ {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [], # templateとして利用するディレクトリのパスを記述
     -- 以下略 --
```

https://docs.djangoproject.com/ja/4.1/ref/settings/#templates

Viewからtemplateに値を渡す

templateを指定する際に、値を渡してtemplateで使用する

Viewの記述

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

valueにHELLOを格納して、templateで表示する

templateの記述

```
<h1>{{ value }}</h1>
```

Django template language(DTL)とは?

HTMLの中にpythonコードを埋め込むためのツール

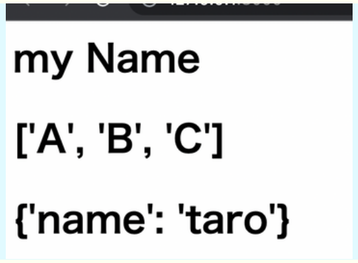
Viewの記述

```
def index(request):
    my_name = "my Name" # 文字列
    letters = ["A", "B", "C"] # リスト
    human_dic = {'name': 'taro'} # 辞書
    return render(request, 'base.html', context={
        'my_name': my_name, 'letters': letters, 'human_dic': human_dic
    })
```

templateの記述

```
<h1>{{ my_name }}</h1> {# 変数 #}
<h1>{{ letters }}</h1> {# リスト型 #}
<h1>{{ human_dic }}</h1> {# 辞書型 #}
```

画面表示



https://docs.djangoproject.com/ja/4.1/ref/templates/language/

DTLの制御文

処理の記述方法

{% %}: 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 %}
```

if文

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

コメント文

```
{# コメント文は画面上に表示されません #}
```

DTLの継承

他のテンプレートを継承して、利用できます。元に、jquery, bootstrapなど全ページ共通のライブラリの読込み、ヘッダーの作成など、を実装します。

元のファイル(base.html)

```
<html>
{% block content %}
    ここの内容は読み込み先が記載する
{% endblock %}
</html>
```

contentの中の内容を継承したファイルで記述することで、処理を共通化できる

継承したファイル

```
{% extend “base.html” %}
{% block content %}
    ここに書きたいことを書く
{% endblock %}
```

テンプレートファイル構成

```
templates/
    base.html
    home.html
    index.html
    list.html
```

Templateフィルター

Viewで渡した値を、Template上で変換する。
Templateでもpythonの関数で利用できる。

文章全体のフィルター {% filter upper %} ~ {% endfilter %}

変数へのフィルター {{ variable | filter }}

add →listに値を追加	floatformat →小数点以下を丸める	lower →小文字に変換	urlencode →入力値をURLとして使えるように エンコードする
capfirst →最初の文字を大文字にする	join →配列を指定した文字を挟んで結ぶ	random →シーケンスからランダムに要素を 取り出す	urlizetrunce →urlを遷移可能にし、指定以上を切り詰める
cut →指定した値を取り除く	last →リストの最後の要素を取り出す	slice →リストの一部を返す	urlize →文字列の中のurlを遷移できるようにする
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	truncatechars →文字を途中で切り詰めてのころを .....にする	linebreaksbr →改行をbrにする
first →リスト中の最初の要素	linebreaks →改行をbrにしpタグを付与する変換 する	upper →大文字に変換する	

https://docs.djangoproject.com/ja/4.1/ref/templates/builtins/#built-in-filter-reference

Templateフィルターの自作

DTLでは、template上でpythonの関数を実行できる。

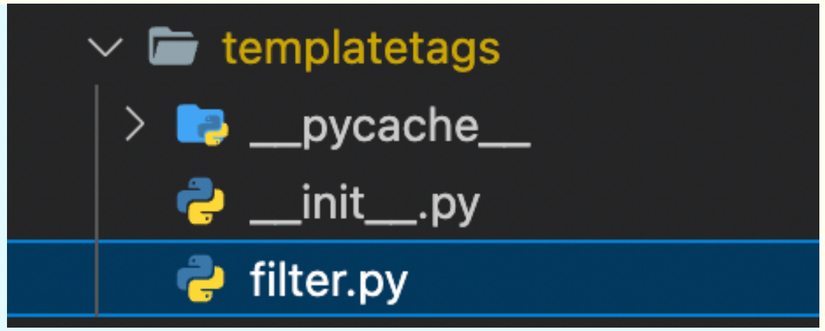
1.アプリケーションフォルダ内にtemplatetagsフォルダを作成

2.templatetagsの中に__init__.pyを作成

3.filterを自作するファイルを作成し、中に関数を記述して、registerを行う

```
from django import template
register = template.Library()

@register.filter(name='my_filter')
def 関数名(value):
```



4.templateでloadして利用する

```
{% load filterのファイル名 %}
{{ value|my_filter }}
```

画面遷移

Templateでの画面遷移方法

{% url %}を用いて、urls.pyで指定した遷移先に移動する

アプリケーションフォルダのurls.py

```
from django.urls import path

app_name = 'app'
urlpatterns = [
    path('home', views.home, name='home'),
]
```

"{% url 'app:home' %}"をaタグのhrefに記述すると
app_nameがapp、nameがhomeのViewに遷移する

Templateの記述

```
<a href="{% url 'app:home' %}">Home</a>
```

Viewに値を渡す場合

```
<a href="{% url 'app:home' val1='val1' val2='val2' %}">Link</a>
```

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す。

Templateでの静的コンテンツの読み込み

```
{% 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
```

静的コンテンツ配置先の設定

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する。

```
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る。

ViewからTemplateにインスタンスを渡す

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

Templateでの取り出し

```
{{ instance.propery1 }}
```

問題

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。