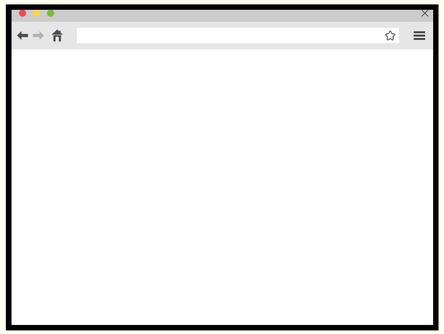
Flaskの基本的な仕組み



HTTP(s)リクエスト

HTTP(s)レスポンス
（HTML形式のテキストを返す）

Flask

http://〇〇/route

View関数

Routeの設定と
URLごとの処理を定義

Model

DBへのアクセス

Template

HTMLの定義

Template

動的なWebページを生成するためにPythonコードとHTMLを組み合わせる機能です。

テンプレートフォルダの作成

デフォルトではFlaskは同じ階層のtemplatesフォルダを見る

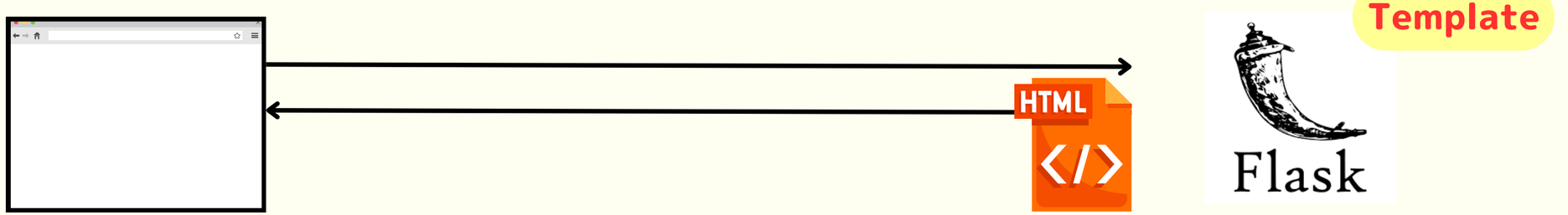
app.py **# Flask実行ファイル（flask=Flask(__init__)を定義する）**
/templates
　/hello.html

Viewからテンプレートフォルダの利用

```
from flask import render_template

def index():
    return render_template('base.html')
```

app = Flask(__name__, **template_folder='フォルダ名'**) # テンプレートを格納するフォルダを設定



Jinja

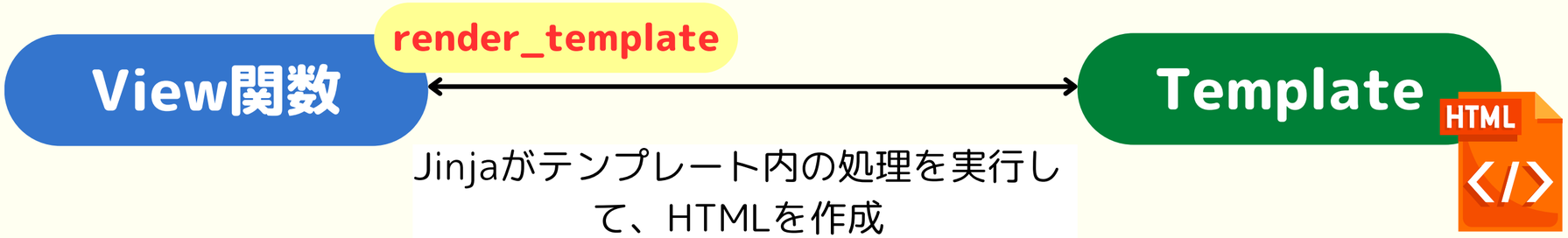
Pythonのテンプレートエンジン、HTML内でPythonを利用するために使われる

Viewから変数を渡す（render_templateでテンプレートを元に動的にレンダリング）

```
def index():
    my_name = "my Name"
    letters = list(my_name)
    human_dic = {'name': 'taro'}
    return render_template('base.html', myvariable=my_name, letters=letters, human_dic=human_dic)
```

テンプレートファイルで変数を表示

```
<h1>{{myvariable}}</h1>: my_name変数の値(my Name)を表示
<h1>{{letters}}</h1>: lettersリストの内容を表示
<h1>{{human_dic['name']}}</h1>: human_dic内のnameキーの値を表示
<h1>{{human_dic.name}}</h1>: human_dic内のプロパティを取得
```



クラスに定義したメソッドを実行

Flaskのビューから渡されたオブジェクトのメソッドは、Jinjaテンプレート内で実行可能

オブジェクト定義例

```
class User:
    def __init__(self, name):
        self.name = name

    def greet(self, greeting):
        return f"{greeting}, {self.name}!"
```

ビュー関数での使用例

```
@app.route('/')
def index():
    user = User("Alice")
    return render_template("index.html", user=user)
```

HTMLテンプレート内でのメソッド呼び出し

```
<h1>{{ user.greet('Hello') }}</h1>
```

Jinjaの制御文

Jinjaテンプレートでは、制御構文を使って動的なHTMLコンテンツを生成できる

1.制御フローの構文 ({% %}): if文やfor文などの制御フローを記述するために使用する
2.値のアウトプット ({{ }}): 変数や式の値をHTMLに表示するために使用する
3.コメント ({# #}): テンプレート内にコメントを記述するために使用する。これは画面上には表示されない

for文の使用例

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

if文の使用例

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

コメント文の使用

```
{# この部分はコメントです #}
```

ループ内の特別な変数

loop.index:
- ループの現在の繰り返しを示す。1から始まる（例: 1, 2, 3, ...）

loop.revindex:
- ループの終わりからの繰り返し回数を示す。1から始まる（例: 3, 2, 1, ...）

loop.first:
- 現在の繰り返しが最初のものであればTrue

loop.last:
- 現在の繰り返しが最後のものであればTrue

loop.length:
- シーケンス内のアイテムの総数を示す

loop.depth:
- 現在の再帰的ループの深さを示す。レベル1から始まる

loop.previtem:
- 前の繰り返しでのアイテムを示す。最初の繰り返しでは定義されていない。

loop.nextitem:
- 次の繰り返しでのアイテムを示す。最後の繰り返しでは定義されていない。

テンプレート継承の基本

共通の要素を持つ「基本テンプレート」を作成し、子テンプレートで個別の要素をオーバーライドする

基本テンプレート（Base Template）

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

子テンプレート（Child Template）

- {% extends %}タグを使用して、他のテンプレートを拡張します。
- 子テンプレートは親テンプレートのブロックをオーバーライドして内容を提供します。

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }} {# 継承先の定義を使用#}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome to my awesome homepage.
    </p>
{% endblock %}
```

テンプレート内のフィルタの使用

Jinjaテンプレート内でデータの表示を変換する

個別の文章にフィルタ

- 変数の後にパイプ記号 (|) とフィルタ名を記述して、その変数にフィルタを適用します。
- 例: {{ variable | upper }} は、variableの値を大文字に変換します。

文章全体にフィルタを適用

- フィルタを適用したいテキストを {% filter [フィルタ名] %} と {% endfilter %} の間に配置
- 例: {% filter upper %}
- このテキストは大文字になります
- {% endfilter %}

abs() →絶対値	float() →数値を浮動小数点数に変換	max() →最大値	trim() →両端の文字を削除（デフォルトは空白）
attr() →属性を返す	format() →formatを利用する	min() →最小値	tojson() →json形式に変換
capitalize() →最初の文字は大文字	groupby() →指定したキーで値をまとめる	random() →シーケンスからランダムに要素を取り出す	upper() →大文字に変換
default() →デフォルト値を指定	int() →数値型に変換する	replace() →文字列を返還する	urlencode() →urlエンコード
dictsort() →辞書型をキーで昇順に入れ替える	join() →配列を指定した文字を挟んで結ぶ	reverse() →リストなどの順番を逆にする	urlize() →文字列の中にあるurlが遷移できるようになる
escape() →エスケープする	last() →リストの最後の要素を取り出す	round() →四捨五入、ceil: 切り上げ、floor: 切り下げ	wordcount() →文字数をカウント
filesizeformat() →kB, MBなどわかやすいファイルのフォーマット	length() →要素の数を返す	sort() →順番を昇順に入れ替える	lower() →小文字に変換する
first() →リストの最初の要素を返す	list() →値をリストに変換する	sum() →シーケンスの合計値	title() →タイトルケースに変換

代表的なフィルター一覧。詳細は下記
https://jinja.palletsprojects.com/en/latest/templates/#builtin-filters

カスタムフィルタの作成

カスタムフィルタの定義

- Flaskアプリケーション内でカスタムフィルタを定義する
- @app.template_filter()デコレータを使用して、新しいフィルタを登録する
- app.add_template_filter(フィルタ関数, 'フィルタ名')

```
from flask import Flask

app = Flask(__name__)

@app.template_filter('my_filter')
def my_filter_function(s):
    return f'xxx{s}xxx'
```

```
app.add_template_filter(my_filter_function, 'my_filter')
```

カスタムフィルタの使用

- HTMLテンプレート内で新しいフィルタを使用する
- パイプ記号 (|) を使用して変数にフィルタを適用する

```
{{ variable | my_filter }}
```

Flaskにおける画面遷移

url_forでリンク作成

- url_for関数を使用して、エンドポイントに基づくURLを動的に生成する
- エンドポイントは通常、関連するビュー関数の名前

```
<a href="{{ url_for('関数名') }}">新しいページへ</a>

<img src="{{ url_for('static', filename='〇〇.jpg') }}">
```

リダイレクトの実装

- redirect関数を使用して、別のURLにクライアントをリダイレクトする
- これは通常、フォーム送信後や特定のアクション後に使用される

```
return redirect(url_for('info', variable='man'))
```

Flaskにおけるエラーハンドリング

エラーハンドラの定義

- @app.errorhandlerデコレータを使用して、特定のHTTPエラーコードに対するカスタムハンドラを定義
- これにより、特定のエラーが発生した際にカスタム応答を提供できる

```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404
```

エラーの強制発生

- abort関数を使用して、任意のHTTPエラーコードを発生させる
- これは、特定の条件下でアクセスを制限する際に役立つ

```
abort(404)
```

演習: ホームページの作成

- **目的**: テニス部の活動、メンバー、規約を紹介するウェブサイトの開発
- **ページ構成**:
  - **メインページ**: シンプルなメッセージ「メインページです」と表示
  - **メンバーー覧**: 太郎さん(21)、次郎さん(20)、良子さん(22)、花子さん(21)の情報をリストアップ。各メンバーの詳細ページへのリンクを含む
  - **メンバー詳細ページ**: 選択されたメンバーの詳細情報と画像（Udemy提供）を赤色のテキストで表示
  - **利用規約ページ**: 「利用規約です」というテキストのみを表示
- **機能**:
  - ヘッダーから各ページへのナビゲーション
  - 誤ったURLへのアクセス時はメインページへリダイレクト