# Python基礎オンライン講座
# 3回目

2025/10/14

Kazuma Sekiguchi

# 前回のAgenda

- テンプレートエンジンを利用する
- Jinja2を利用したテンプレートの利用
- テンプレートでの分岐処理
- テンプレートでの繰り返し
- Jinja2のフィルター利用

# 今回のAgenda

- フォームデータの受け取り
- WTFormsを利用したフォームの作成
- バリデーションを行なう
- セッションを利用する

# Flaskでフォームを利用する

- HTTPでデータを送信する方法は2種類存在する
  - GETメソッド
  - POSTメソッド
- GETの場合は、URLとしてデータを取得するときに使われる
  - 同時クエリパラメータというパラメータを送出することが可能
- POSTの場合は、フォームなどを利用してデータを送信するときに利用する
  - ファイルなども送信可能になる
  - 比較的大量のデータを送信するときに使う

# Flaskでフォームを利用する

- Flaskを使うことで、GETでもPOSTでもデータ受信は可能
  - requestをimportしておく必要はある
- リクエストメソッド（POSTかGETなど）、リクエストパラメータ（クエリパラメータ）、リクエストボディなどを取得可能
  - POSTで送られるフォームデータはリクエストボディから取得できる
  - GETはrequest.args.getからリクエストパラメータを取得可能
  - POSTはrequest.form.getからリクエストボディを取得可能

# WTFormsライブラリの利用

- WTFormsを利用すると、入力値の検証（バリデーション）やセキュリティ対策を行なうことが可能
  - WTFormsライブラリはpipでインストールが必要

```
pip install wtforms==3.1.2
```

今回は一応バージョン指定する

- WTFormsを使うことで、入力フィールドを作成することが可能
  - HTMLでも作成できるがWebアプリとして利用する場合は、エラー処理などもあるため、WTFormsを経由してフォームを作成した方が楽

# Formクラスとして作成

- 入力フィールドを作成していく場合は、クラスとして作成する
  - app.pyファイルで読み込んで表示するようにする

```
name = StringField('名前:',render_kw={"placeholder":"山田太郎"})
age = IntegerField('年齢:',default=20)
```

| フィールドクラス | 作成されるフィールド |
|---|---|
| StringField | 文字列入力フィールド |
| IntegerField | 整数入力フィールド |
| BooleanField | 真偽値入力フィールド |
| RadioField | ラジオボタン入力フィールド |
| SelectField | ドロップダウン入力フィールド |
| TextAreaField | 複数行入力フィールド |
| DateField | 日付け入力フィールド |
| PasswordField | パスワード入力フィールド |
| EmailField | メールアドレス入力フィールド |
| HiddenField | 隠し入力フィールド |
| SubmitField | 送信ボタンフィールド |

# formのデータを渡してフォームを表示

- HTMLファイルに対して、formデータを渡して表示を行なう
  - 入力フィールドの引数に見た目やサイズなどを指定することが可能

```
<form method="POST” novalidate>
{{form.name.label}}{{form.name(size=20)}}<br>
{{form.age.label}}{{form.age()}}<br>
{{form.password.label}}{{form.password(size=20)}}<br>
```

| 引数 | 内容 |
|---|---|
| size | 入力フィールドの文字数サイズ |
| maxlength | 入力フィールドの最大入力文字数 |
| disabled | 入力フィールドを無効にする |
| readonly | 入力フィールドを読み取り専用にする |
| placeholder | 入力フィールドに表示するヒント |
| style | 入力フィールドの大きさを指定 |

# WTFormsでバリデーションを行なう

- WTFormsでバリデーションを行なうことが可能
  - validatorをインポートすることで利用することが可能
  - 使うバリデーションごとに読み込む指定が必要

| バリデータクラス | 作成されるバリデーション |
| --- | --- |
| DataRequired | 入力必須 |
| Length | 文字列の長さ指定 |
| Email | メールアドレス形式 |
| URL | URL形式 |
| Regexp | 正規表現による検証 |
| NumberRange | 数値の範囲指定 |
| Optional | 入力が無くても問題ない |
| EqualTo | フィールド同士を比較して一致する |

# バリデーションの利用方法

```
name = StringField('名前:',validators=[DataRequired('名前は必須です
')],render_kw={"placeholder":"山田太郎"})
age = IntegerField('年齢:',validators=[NumberRange(18,100,'入力範囲は18歳から
100歳までです')],default=20)
password = PasswordField('パスワード:',validators=[Length(8,80,'パスワードの長さは
8文字以上80文字以下です'),EqualTo('confirm_password','パスワードが一致しませ
ん')])
```

- validatorsでバリデーションを指定する
  - バリデーションルールは複数指定することも可能
- バリデーションが問題なければ、form.validate()がtrueになる

# メールアドレスのチェック

- メールアドレスのチェックをするときは、別途モジュールが必要

```
pip install email-validator==2.2.0
```

今回は一応バージョン指定する

# HTMLでのチェックを無効にして確認

- WTFormsでフォームを指定すると、HTMLのチェック機能が有効になる
  - HTMLでのバリデーションが動作する
  - ユーザーによって回避ができてしまうため、確実なものではない
  - Python側のチェックを利用することが必須
- 確認するときには邪魔なので、HTMLのチェック機能を解除する
  - novalidate属性をformタグに追加して無効にする

# Flask-WTFを利用する

- Flask-WTFを利用するとバリデーション、フィールドの自動生成やCSRF対策がより簡単に対応することが可能

```
pip install flask-wtf==1.2.2
```

- Flask-WTFはWTFormsの機能をFlaskに統合するための拡張機能
- CSRF脆弱性などの対応が可能なため利用した方が無難
- 乱数を作成して設定しておくことで、CSRF脆弱性に対応できるほか、セッションを利用することが可能

# CSRF脆弱性の対策

- CSRF（クロスサイトリクエストフォージェリ）攻撃とも呼ぶ
  - 悪意のあるサイトがユーザーのブラウザを使って別のサイトに不正なリクエストを送信する攻撃

- Flask-WTFのCSRF保護機能
  - hidden_tag()メソッドでCSRFトークンを自動生成
  - トークンはセッションに保存され、リクエスト時に一致を確認
  - フォーム送信時にCSRFトークンが含まれ、サーバー側で検証される
    - もしトークンが一致していない場合は、正常なアクセスをしていないとしてエラー扱いにする

# セッション

- セッションを使うとユーザーが入力したデータを再利用することが可能
  - Flaskのセッションはサーバー側保存されない点に注意
  - Cookie内に暗号化された状態で保存される
- 辞書型のデータとして格納される

```
session['user'] = 'Taro' #セッションへの値の保存
session.get('user') #セッションから指定したデータを取得（存在しない場合はNone）
session.pop('user', None) #指定したキーを削除
session.clear() #全削除
```

- app.secret_keyの設定が必要

# flush

- メッセージをセッションに一時的に保存し、次のHTTPリクエストで取り出して表示したら自動的に消える仕組み
  - 内部的には、session[‘_flashes’]というキーにデータを格納している
  - 1回だけ表示したい、ときに利用するのが最適

```
flash('Message sent successfully!', 'success')
```

第一引数が表示するメッセージ内容
第二引数はカテゴリ（メッセージの種類）

  - 表示するときは、テンプレート側で

```
get_flashed_messages(with_categories=True)
```

と記述する

  - with_categories=Trueを記述するとカテゴリ付きで取得する

# 組み込んだ例

```
from flask import Flask, render_template, request,flush,session
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Length

app = Flask(__name__)
app.secret_key = 'secret_key'

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=25)])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        # フォームのデータが有効な場合の処理
        return "Logged in"
    return render_template('login.html', form=form)

if __name__ == '__main__':
    app.run()
```

必要なモジュールのインポート

乱数の設定

フィールドの作成

フォームのバリデーションを通った場合

最初の表示またはバリデーションでエラーになった場合

# 組み込んだ例

```
<form method="POST">
    {{ form.hidden_tag() }}
    <div>
        {{ form.username.label }}<br>
        {{ form.username(size=20) }}<br>
        {% for error in form.username.errors %}
            <span style="color: red;">{{ error }}</span><br>
        {% endfor %}
    </div>
    <div>
        {{ form.password.label }}<br>
        {{ form.password(size=20) }}<br>
        {% for error in form.password.errors %}
            <span style="color: red;">{{ error }}</span><br>
        {% endfor %}
    </div>
    <div>
        {{ form.submit() }}
    </div>
</form>
```

CSRF対応のフィールドが記述される

バリデーションでエラーが生じたときの処理
username.errorsにエラー内容が格納される
エラー内容が複数ある場合があるため、ループで処理を行なう

バリデーションでエラーが生じたときの処理

送信ボタンの表示

ありがとうございました。

また次回。