

Flask(Form(リクエストの取得))

```
from flask import request
```

request: ユーザーが送信したリクエストにアクセスする。

GETリクエストの場合

```
var1 = request.args.get('var1')
```

```
var2 = request.args.get('var2')
```

POSTリクエストの場合

```
var1 = request.form.get('var1')
```

```
var2 = request.form.get('var2')
```

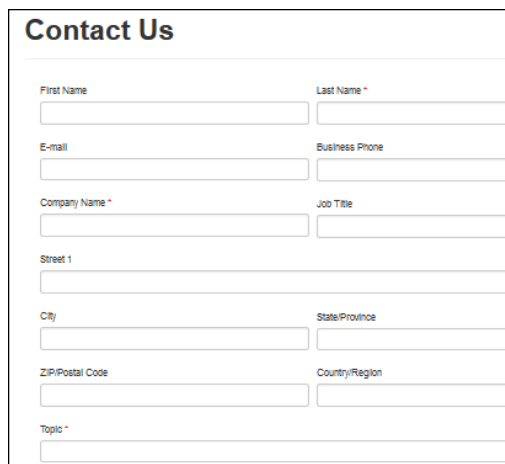
GETかPOSTか確認する

```
request.method # GET or POST
```

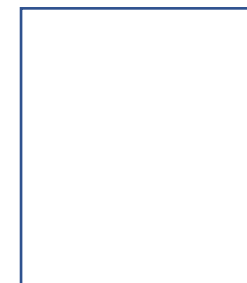
関数に対応するリクエストを指定(デフォルトはGETのみ)

```
@app.route('/', methods=['GET', 'POST'])
```

フォームを入力
(Template)



The image shows a web form titled "Contact Us". It contains several input fields: "First Name", "Last Name" (marked with a red asterisk), "E-mail", "Business Phone", "Company Name" (marked with a red asterisk), "Job Title", "Street 1", "City", "State/Province", "ZIP/Postal Code", "Country/Region", and "Topic" (marked with a red asterisk). Each field is represented by a text input box.



入力された値を取り出す
(View)

Flask(form(画像のアップロード))

Flaskで画像データのアップロード方法について説明します

html側

```
<form method="POST" enctype="multipart/form-data">  
  <input type="file" name="file">  
  <input type="submit" value="アップロード">  
</form>
```

Python側

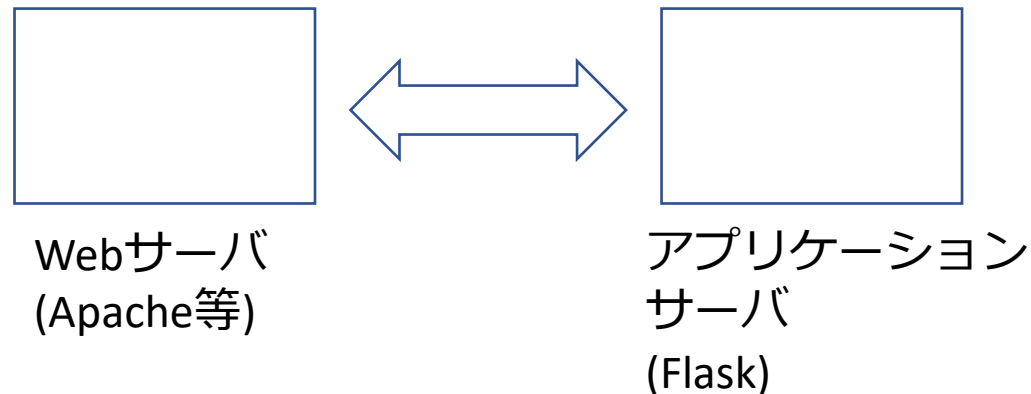
```
file = request.files['file']  
save_filename = secure_filename(file.filename)  
file.save(file_path)
```

secure_filename(from werkzeug.utils import secure_filename): ファイル名を適切な形に変換する処理。
(例: .././a.jpgのような名前のファイルだと誤った場所に保存される)

*) ただし、secure_filenameは日本語文字に対応していないため、別途日本語を英語に変換するライブラリが必要
以下をインストールする

```
pip install wtforms  
pip install pykakasi
```

werkzeug(ヴェルクツォイク)・・・WSGIユーティリティライブラリ。WSGIはWeb Server Gateway InterfaceでWebサーバとアプリケーションサーバ間の接続をするためのインタフェースです



Flask(wtform)

```
from wtforms.form import Form
from wtforms import StringField, SubmitField
```

セキュリティキーの設定

app.config['SECRET_KEY'] = 'secretkey'
セッションなどで用いられるセキュリティ上の
問題で利用される。できるだけランダムな値を設定する
のが望ましく、以下のコマンドで作成した値をコピー
するとよい

```
import os; print(os.urandom(16))
```

python(Formの作成)

```
class InfoForm(Form):
    name = StringField('名前(は ?)')
    submit = SubmitField('Submit')

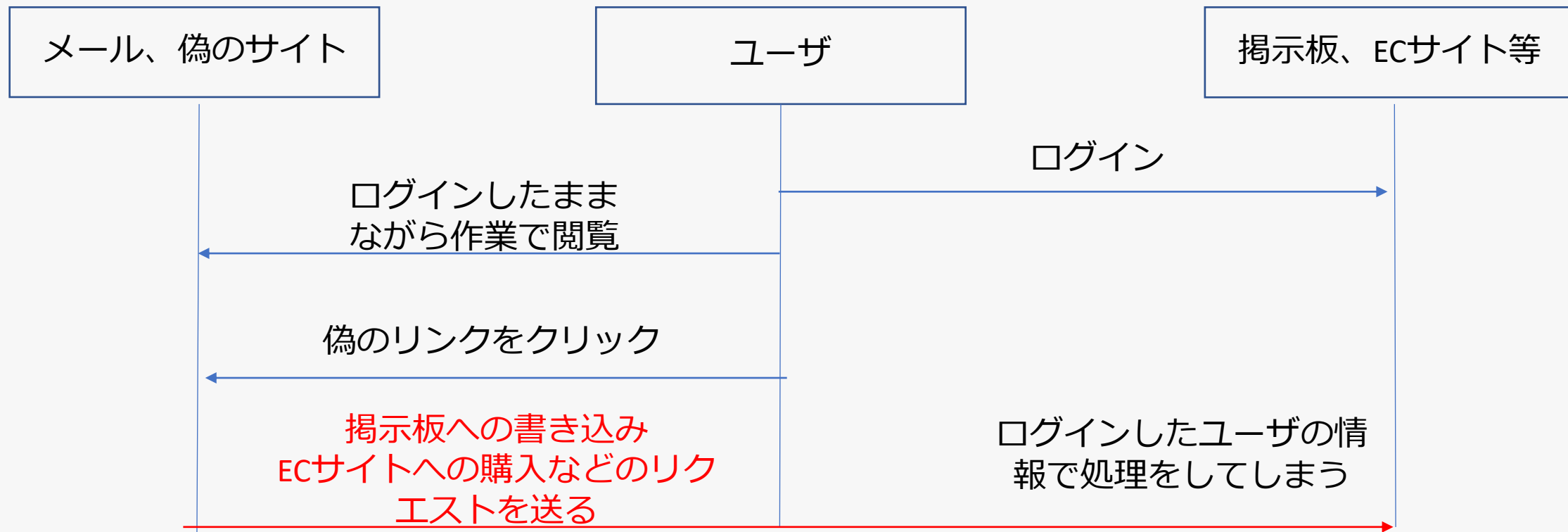
    {{ form.csrf_token }} <!-- csrf対策-->
    <!-- Textフィールドを作成-->
    {{ form.name.label }}{{ form.name() }}
    {{ form.submit() }}
```

FormのField一覧

| | |
|---------------------|-----------------------------|
| StringField | 文字列の入力(<input type='text'>) |
| TextAreaField | テキストエリア(<textarea>) |
| DateField | 日付を扱う |
| DateTimeField | タイムスタンプを扱う |
| BooleanField | 真偽値を扱う(checkbox) |
| FileField | アップロードするファイルを扱う |
| DecimalField | Decimal型を扱う |
| FloatField | 浮動小数点数を扱う |
| IntegerField | 数値型を扱う |
| RadioField | ラジオボタンを扱う |
| SelectMultipleField | セレクトボックスを扱う |
| FormField | 他のフォームを埋め込んで利用する |
| HiddenField | hidden要素を扱う |
| PasswordField | パスワードを扱う |
| SubmitField | 送信ボタンを扱う |

攻撃手法(クロスサイトリクエストフォージェリ(CSRF))

メールやSNS、他のサイトを通じて、標的となるページに強制的にリクエストを送らせて、掲示板の書き込み、オンラインショッピングでの注文をさせる



対策・・・重要な処理の場合にはパスワードを要求させる。**csrf_token**を利用する(乱数文字列で対象の掲示板、ECサイトでの遷移しか処理を受け付けないようにする仕組み)。**CAPTCHA**を利用してプログラムでの処理でなく人間の入力であることを保証する

Flask(formの続き、 templateの関数インポート、 セッション)

関数のインポート

インポート元(_form.html)

```
{% macro function(field) %}
```

~

```
{% endmacro %}
```

インポート先(関数functionをインポートする)

```
{% from "_form.html" import function %}
```

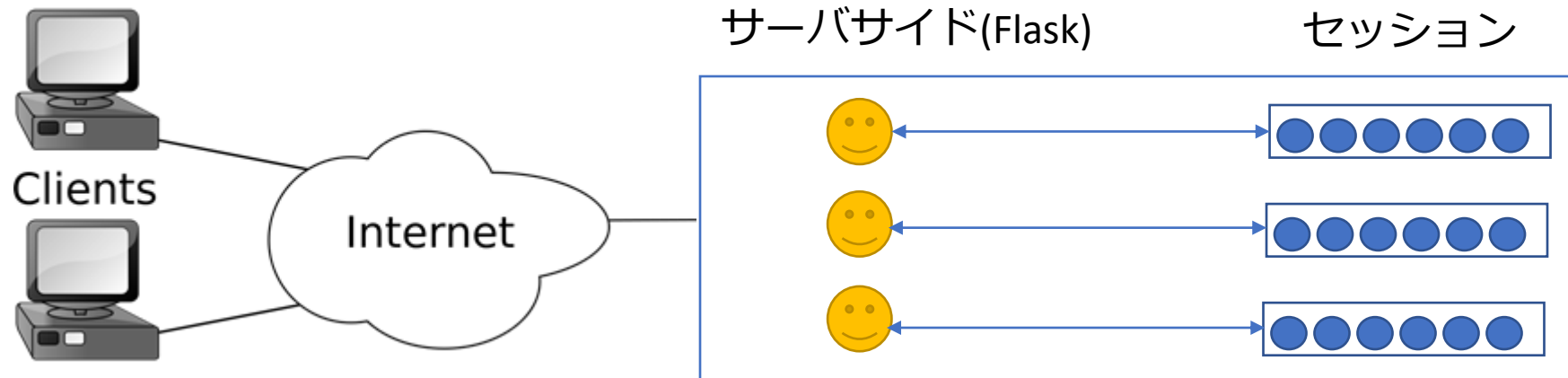
formの場合以下のrender_field関数を継承して利用すると便利
(詳細は公式:<https://flask.palletsprojects.com/en/1.1.x/patterns/wtforms/>)

```
{% macro render_field(field) %}  
<dt>{{ field.label }}  
<dd>{{ field(**kwargs)|safe }}  
{% if field.errors %}  
<ul class=errors>  
{% for error in field.errors %}  
<li>{{ error }}</li>  
{% endfor %}  
</ul>  
{% endif %}  
</dd>  
{% endmacro %}
```

セッション

サーバのメモリ上にユーザに応じた情報を保存して管理する

Flaskでは、flask.sessionというセッション用のライブラリが存在する



Flask(fieldについて詳細)

公式: <https://wtforms.readthedocs.io/en/latest/fields/>

テキストボックスの長さの変更

```
{{ form.field(size=〇〇) }}  
{{ render_field(form.field, size=12) }}
```

デフォルト値

画面に入力するデフォルトの値を設定する

```
name = StringField('名前: ', default='Flask太郎')
```

チェックボックスでデフォルトはチェックする

```
{{ form.field(checked=True) }}  
{{ render_field(form.field, checked=True) }}
```

プレースホルダー

ユーザが入力しやすくなるように入力フォーム上に仮の値を薄く表示させること
オプションとして、render_kw={"placeholder": "yyyy/mm/dd"}を追加する

DateFieldにプレースホルダーを追加する

```
birthday = DateField('誕生日: ', format='%Y/%m/%d', render_kw={"placeholder": "yyyy/mm/dd"})
```

誕生日:

クラス追加

```
{{ form.field(class=〇〇) }}  
{{ render_field(form.field, class="class-name") }}
```

widgetの変更

```
name = StringField('名前: ', widget=TextArea()) # textからtextareaに変更
```

```
import wtforms.widgets
```

(公式: <https://wtforms.readthedocs.io/en/2.2.1/widgets/>)

Flask(バリデーション)

バリデーションについて詳細(<https://wtforms.readthedocs.io/en/latest/validators/>)

バリデーションチェック

```
from wtforms.validators import DataRequired
```

fieldで以下のように記載する

```
StringField(validators=[DataRequired('データを入力してください')])
```

単体のバリデーターの自作

以下の関数を作成してフィールドごとにバリデーションをする

```
def validate_フィールド名(form, field):
```

```
    pass
```

または、自作関数をvalidatorsに指定する

```
validators=[自作関数]
```

複数のフィールドをバリデーションしたい場合(validate関数の上書き)

```
def validate(self):
```

```
    if not super(Form, self).validate():
```

```
        return False
```

```
    ..バリデーションを追加（問題がある場合はFalseを返すようにする）
```

フラッシュメッセージの表示(flask.flash)

```
flash('メッセージ'), {% for message in get_flashed_messages() %} {{ message }} {% endfor %}
```

代表的なバリデータ

| | |
|--------------|--|
| DataRequired | データが入っていない場合にエラーとして値を返す |
| Email | メール型でないと入力できなくする(インストールが必要(2020/5/22時点)。 pip install wtforms[email]) |
| EqualTo | 他のフィールドと等しいか確認する。 |
| Length | 文字列の長さを指定する |
| NumberRange | 数値の大きさの範囲を指定する |