Python Web Development

powered by Flask

파이썬 웹 프레임워크

- Flask
- Django
- Pyramid
- Bottle
- ...

파이썬 웹 프레임워크

- Flask ⬅
- Django
- Pyramid
- Bottle
- ...

Hello, Flask!

가장 간단한 플라스크 웹 어플리케이션을 만들어 보자.
`helloflask.py`를 만들고 다음의 코드를 입력한다:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello, Flask!'
```

서브라임 텍스트에서 코드를 실행해도 아무 일도 일어나지 않는다. 플라스크 어플리케이션을 실행하는 방법은 후에 설명한다.

먼저 코드를 한 부분씩 살펴보자.

```
from flask import Flask
```

플라스크 모듈(`flask`)에서 플라스크 클래스(`Flask`)를 임포트한다.

```
app = Flask(__name__)
```

`Flask` 클래스를 이용하여 새로운 플라스크 인스턴스를 만든다. 이렇게 만들어진 플라스크 인스턴스는 그 자체로 WSGI 어플리케이션이 된다. `Flask` 클래스의 첫 번째 인자로는 항상 `__name__`을 넘겨주도록 한다.

여기까지가 플라스크 어플리케이션에 기본적으로 들어가는 코드들이다.

```
@app.route('/')
def index():
        return 'Hello, Flask!'
```

`app.route` 데코레이터를 이용해 루트 URL(`/`)이 `index` 함수를 호출해야 한다고 알려준다. `index`는 `'Hello, Flask!'`라는 문자열을 반환하는데, 플라스크는 이를 자동으로 HTTP 응답으로 변환하여 클라이언트에게 전달한다.

만약 우리의 웹 사이트 주소가 `http://example.com` 이라면 `http://example.com/`으로 들어오는 HTTP 요청을 `index` 함수가 처리하도록 하는 것이다.

이제 작성한 코드를 실행해 볼 차례이다. 터미널(혹은 명령 프롬프트)에서 다음을 그대로 입력한다(`helloflask.py`가 있는 디렉토리로 이동한 상태여야 한다):

```
$ export FLASK_APP=helloflask.py
$ export FLASK_DEBUG=1
$ flask run --reload
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
...
```

(윈도우 유저는 `export` 대신 `set`을 사용한다)

브라우저에서 http://127.0.0.1:5000/ 으로 접속하여 제대로 실행되는지 확인한다.

URL 라우팅

플라스크의 URL 라우팅 규칙은 간단하다. `helloflask.py`의 5-7번 줄을 지우고 다음의 코드를 입력한다:

```
@app.route('/hello):
def hello():
    return 'Hello'

@app.route('/world'):
def world():
    return 'World'
```

코드를 모두 작성하고 저장하면 또다시 `$ flask run --reload`를 입력하지 않아도 `--reload` 옵션 덕분에 Flask가 변경 사항을 감지하고 자동으로 서버를 다시 시작한다. 만약 코드에 오류가 있어 Flask 서버가 중지된 경우에는 다시 실행해야 한다.

브라우저에서 http://127.0.0.1:5000/hello, http://127.0.0.1:5000/world 로 접속해 결과가 예상한 바와 같은지 확인한다.

컨버터

Flask의 URL 라우팅 시스템은 `/hello`, `/world`와 같은 정적 URL 외에 `/user/12101`, `/page/3` 등의 동적인 URL도 처리할 수 있다.

```
@app.route('/show/<message>')
def show(message):
    return 'You said: %s' % message
```

또한 `<message>` 대신 `<string:message>`의 형식으로도 작성이 가능한데, 여기서 `string`을 `message`의 컨버터라고 한다. 컨버터는 변수를 특정 타입으로 변환(Convert)하는 역할을 한다.

자주 쓰이는 컨버터들은 다음과 같다. 컨버터를 지정하지 않는 경우는 기본 값으로 string 컨버터가 선택된다.

컨버터	설명
string	슬래쉬(`/`)를 제외한 텍스트를 받는다. (기본값)
int	정수를 받는다.
float	부동 소수점 실수를 받는다.
path	string과 비슷하지만 슬래쉬(`/`) 또한 받는다.

템플릿 렌더링

템플릿을 이용해 HTML 페이지를 렌더링 하는 방법에 대해 알아보자.
우선 다음과 같은 코드를 작성하여 실행한다:

```
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
        return render_template('hello.html', name=name)
```

(플라스크 앱에 기본적으로 들어가는 코드들은 생략하였음에 유의한다)

그리고 `hello.html` 파일을 다음과 같은 디렉토리 구조를 따르도록 하여 만든다.

```
/application.py
/templates
    /hello.html
```

`hello.html`의 내용은 다음과 같이 한다:

```
<!doctype html>
{% if name %}
    <h1>Hello, {{ name }}!</h1>
{% else %}
    <h1>Hello, World!</h1>
{% endif %}
```

`hello.html`은 우리가 사용할 템플릿 파일이다. 플라스크의 템플릿 엔진을 이용하면 이 템플릿을 렌더링하여 동적인 웹 페이지를 생성할 수 있다.

플라스크는 기본적으로 Jinja2라는 템플릿 엔진을 내장하고 있으며, Jinja2의 템플릿 문법은 파이썬과 유사하다.

자주 쓰이는 템플릿 문법을 몇 가지 예제를 통해 알아보자.

객체

```
<title>{{ title }} - Flask Application</title>
<body>
    <h1>{{ greeting }}</h1>
</body>
```

객체는 `{{`와 `}}`로 감싸서 나타낸다. 이 객체들은 파이썬 코드에서

```
render_template(..., title='Main', greeting='Hello, World!')
```

와 같이 키워드 인자를 넘겨줌으로써 템플릿 엔진이 인식할 수 있게 된다.

또한 객체가 딕셔너리(`dict`) 타입인 경우 파이썬 스타일의 `dict[key]` 형식의 접근 방식 대신 `dict.key`로 접근할 수 있다. 즉, 다음과 같은 코드를 사용할 수 있다.

파이썬:

```
return render_template(..., user={'name': 'Hanjun', 'job': 'Student'})
```

템플릿:

```
{{ user.name }} is a {{ user.job }}.
```

조건 - if, elif, else, endif

```
{% if name == 'admin' %}
    <h1>Hello, Administrator.</h1>
{% elif name == 'hanjun' %}
    <h1>Hello, Hanjun Kim.</h1>
{% else %}
    <h1>Hello, {{ name }}.</h1>
{% endif %}
```

조건문은 `endif`를 빼면 파이썬과 거의 동일하다.

Jinja2 템플릿 문법에서 조건, 반복 등의 로직과 관련된 부분은 `{%`와 `%}`로 감싼다. 또한 대부분의 경우 닫는 문법(위 예의 경우 `{% endif %}`가 이에 해당)을 필요로 한다.

반복 - for

```
<ul id=user-list>
{% for user in users %}
    <li class=user>{{ user.name }} <span class=age>{{ user.age }}</span></li>
{% endfor %}
</ul>
```

반복문도 파이썬의 반복문과 동일하다.