

마이크로 웹 프레임워크

플라스크(Flask) 1 

# 가상 환경 구축하기

- 프로젝트 별로 가상환경 만들기
  - virtualenv 설치하기(Scripts 디렉토리) & path 등록

```
python -m pip install virtualenv
```
  - 프로젝트 디렉토리에서 가상 환경 만들기

```
c:\myWeb> virtualenv myenv
```
  - 생성된 가상환경 디렉토리내에 activate.bat 실행

```
c:\myWeb\myenv\scripts\activate.bat
```
  - Visual studio에 가상환경 등록
    - Python Environments → Add virtual environment 등록

# Flask

- 파이썬 기반의 마이크로 웹 프레임워크
  - <http://flask.pocoo.org/>
  - “마이크로는 여러분의 웹 애플리케이션이 하나의 파이썬 파일로 개발해야 한다는 걸 말하는게 아니며, 기능적으로 부족하다는 걸 의미하지 않는다.”
  - “마이크로란? 핵심기능만 간결하게 유지하면서, 필요한 기능을 손쉽게 확장할 수 있게 한다는 것을 의미한다”
  - 특징
    - 개발용 서버와 디버거 내장
    - 단위 테스트와 통합 지원
    - RESTful 요청 처리
    - 신사2(Jinja2) 템플릿 엔진 내장
    - 안전한 쿠키 지원
    - WSGI(Web Server Gateway Interface) 호환
    - 유니코드 기반
    - 구글 앱 엔진 호환

# Flask 동작

- 설치하기
- 작성하기

```
python -m pip install flask
```

```
from flask import Flask
#the name of the application package
app = Flask(__name__)
```

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

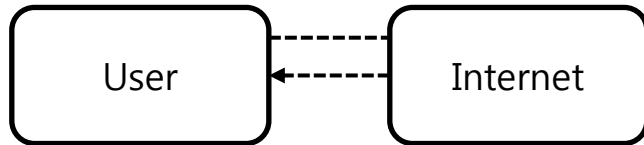
```
if __name__ == '__main__':
    app.run()
```

```
app.debug=True #소스 변경과 디버거 제공
app.run(host='***.***.***.***', port=****)
```

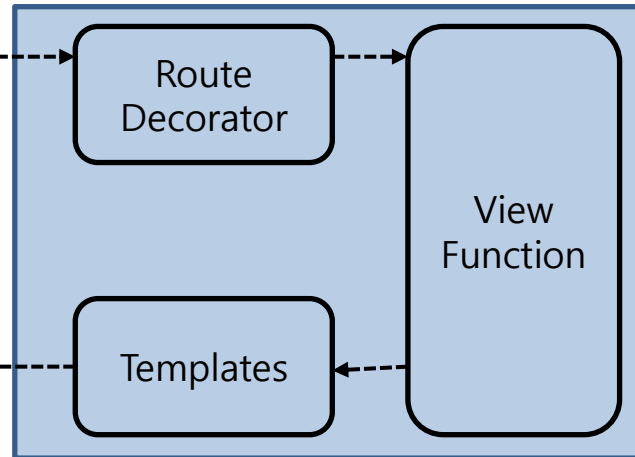
<http://127.0.0.1:5000>  
<http://localhost:5000>

# Flask 실행 과정

1. URL(<http://localhost:5000>) 호출



2. 호출한 URL로 지정된  
뷰 함수 호출



3. 호출한 요청을 분석하여  
비즈니스 로직 실행

- 요청의 상태 유지를 위해 쿠키나 세션 생성
- 오류 발생 시 오류 처리
- 비즈니스 로직의 상태를 위한 로깅

4. 비즈니스 로직의  
결과를 응답으로 전송

5. 응답으로 전송할 값을  
HTML에 표현

```
from flask import Flask
```

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

```
@app.route('/')
```

```
def hello_world():
```

```
    return 'Hello World!'
```

```
if __name__ == '__main__':
```

```
    app.run()
```

# 라우팅

- rout()
  - 복잡한 URL을 쉽게 연결할 수 있는 데코레이터 함수

```
from flask import Flask
```

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

```
@app.route('/hello')
```

```
def hello_world():  
    return 'Hello World!'
```

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

<http://localhost:5000/hello>

# 동적 라우팅

- URL에 동적인 변수 사용하기
  - 원하는 위치에 <변환타입:변수> 추가
    - 디폴트 문자열, int, float 변환 가능

http://localhost:5000/profile/greenjoa

```
@app.route('/profile/<username>')  
def get_profile(username):  
    return 'profile : ' + username
```

http://localhost:5000/message/300

```
@app.route('/message/<int:message_id>')  
def get_message(message_id):  
    return 'message_id : %d' % message_id
```

# URL 와 리디렉션

- URL의 규칙은 아파치 웹 서버나 전통적인 웹 서버의 전례를 따름
- URL 끝의 슬래쉬(/)
  - 파일 시스템의 폴더와 유사하게 처리
  - /가 없이 입력해도 flask가 /를 포함시킨 정규 URL 만듦

```
@app.route('/projects/')  
def projects():  
    return 'The project page'
```

```
@app.route('/about')  
def about():  
    return 'The about page'
```



# url\_for 함수

- 특정함수를 호출하는 URL를 반환해 주는 함수
  - (함수명, URL로 전달 되는 값)

```
from flask import Flask, url_for
```

```
with app.test_request_context():  
    print(url_for('get_profile', username='greenjoa'))
```

- app.test\_request\_context() 함수
  - Flask에서 제공하는 HTTP 요청을 테스트할 수 있는 함수
- redirect(url\_for('get\_profile', username='greenjoa'))
  - redirect 하기 위한 함수

# 요청과 응답

- request
  - http 요청에 관한 정보를 담는 전역객체
    - method : 요청된 http 메서드에 대한 정보
    - args : GET 방식으로 전달된 인자를 참조
    - form : POST 나 PUT 방식의 HTML 폼 데이터 참조

```
@app.route('/profile/', methods=['POST','GET'])
def profile(username=None):
    error=None
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        if not username and not email:
            return add_profile(request.form)
    else:
        error = 'Invalid username or email'
    return render_template('profile.html', error=error)
```

# 정적 파일

- 정적파일

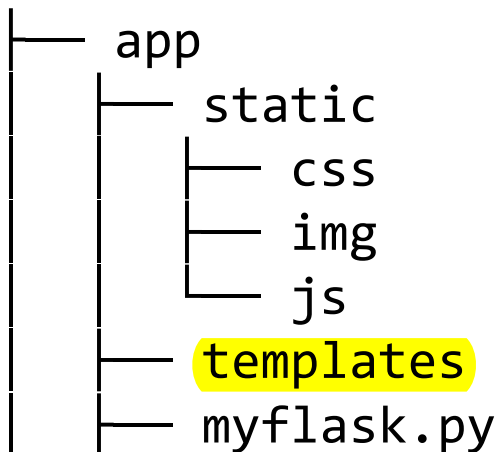
- 자바스크립트, 이미지, CSS

- /static 위치에 정적 파일을 제공해야 함

`url_for('static', filename='style.css')`

함수의 결과 : `/static/css/style.css`

- 구조



# 정적 파일

```
body      { font-family: sans-serif; background: #eee; }
a, h1, h2 { color: #377BA8; }
h1, h2    { font-family: 'Georgia', serif; margin: 0; }
h1        { border-bottom: 2px solid #eee; }
h2        { font-size: 1.2em; }

.page     { margin: 2em auto; width: 35em; border: 5px solid #ccc;
           padding: 0.8em; background: white; }
.entries  { list-style: none; margin: 0; padding: 0; }
.entries li { margin: 0.8em 1.2em; }
.entries li h2 { margin-left: -1em; }
.add-entry { font-size: 0.9em; border-bottom: 1px solid #ccc; }
.add-entry dl { font-weight: bold; }
.metanav   { text-align: right; font-size: 0.8em; padding: 0.3em;
           margin-bottom: 1em; background: #fafafa; }
.flash     { background: #CEE5F5; padding: 0.5em;
           border: 1px solid #AACBE2; }
.error     { background: #F0D6D6; padding: 0.5em; }
```

<head>

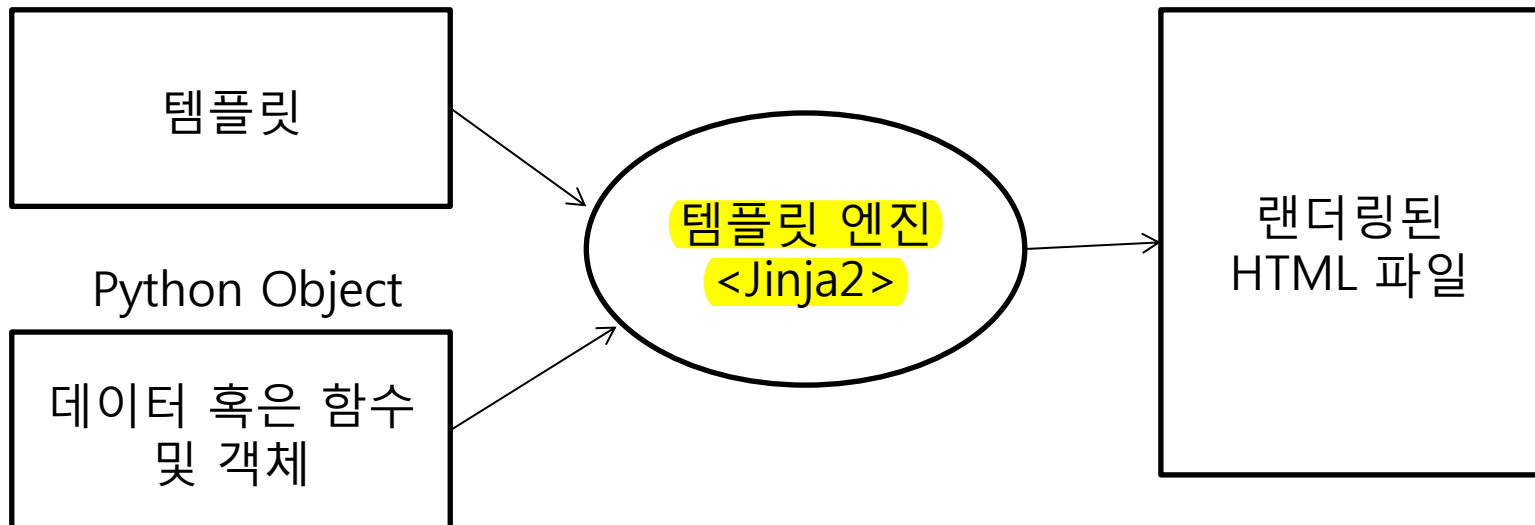
.....

**<link rel="stylesheet" href="{ { url\_for('static',filename='css/style.css') }}" >**

</head>

# 템플릿

- templates
  - 화면 구성에 대한 틀을 의미하는 파일
  - 템플릿 엔진은 데이터를 템플릿에 쉽게 넣을 수 있는 구조를 제공



# 템플릿

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

-. templates 디렉토리  
-. hello.html

```
<!DOCTYPE html>  
  
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">  
<head>  
    <meta charset="utf-8" />  
    <title>greenjoa's homepage</title>  
</head>  
<body>  
    <h1>greenjoa</h1>  
</body>  
</html>
```

# 템플릿

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

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>greenjoa's homepage</title>
</head>
<body>
    <h1>{{name}}</h1>
</body>
</html>
```

**{{ }} : 변수나 표현식의 결과 출력**

# 템플릿

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title> greenjoa's homepage </title>
</head>
<body>
  {% if name %}
    <h1>{{name}}</h1>
  {% else %}
    <h1>Hello World!</h1>
  {% endif %}
</body>
</html>
```

{%    %} : if문, for문 제어문  
{#    #} : 주석



# 템플릿

```
@app.route('/')
def hello_world():
    data={
        'title' : 'Hello',
        'name' : 'greenjoa'
    }
    return render_template('first.html', **data)
```

```
.....
    <title>{{title}}</title>
.....
.....
    <h1>{{name}}</h1>
</body>
```

# 템플릿

- 상속

```
<body>
{% block 블록이름 %}
{% endblock %}
</body>
```

```
<body>
{% block body %}
{% endblock %}
</body>
```

main.html

```
{% extends "hello.html" %}
{% block body %}
It works!!!!
{% endblock %}
```