Flask

플라스크는 파이썬으로 작성된 웹 개발 도구로 마이크로 프레임워크라고도 불린다.
플라스크로 웹 개발을 하는데에 특별한 도구나 라이브러리가 전혀 필요 없기 때문이다.

가상환경 설치

가상환경을 설치하여 패키지를 별도로 관리할 수 있다.

```
$ pip install virtualenv

$ python -m venv work

$ work\Scripts\activate

(work)c:\>
```

간단한 플라스크 앱 만들어보기

```
routes.py

from flask import Flask
app = Flask(__name__)

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

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

> `@(어노테이션)` 이 하는 일 : 웹 브라우저에서 실행 되면 hello_world() 함수를 호출하라는 뜻. 서블릿에 따로 등록을 안시켜도 된다.

- `app = Flask(__name__)`
  가장 먼저 전역에서 Flask 클래스 객체를 선언하였다.

해당 app 객체를 통해 플라스크를 사용할 수 있게 된다.
`__name__`의 경우, 해당 코드를 직접 실행시킬 경우, `"__main__"`이라는 문자열이 들

- `@app.route('/')`
  데코레이터를 사용하여 hello_world 함수를 wrapping 되었음을 알수 있다.
  이 데코레이터를 통해 웹으로 접근한 **사용자가 어떤 URL에 따라 어떤 함수를 실행시켜야 할지 정해주거나**
  위의 예제의 경우 '/'(즉, 메인페이지)가 URL 파라미터인 셈이다.
- `app.run()`
  app 의 run() 메소드를 통하여 Flask 웹 어플리케이션을 실행할 수 있다.
  아무런 인자도 넘겨주지 않을 경우,
  적당히 포트를 잡고 127.0.0.1 로 실행시키게 된다.

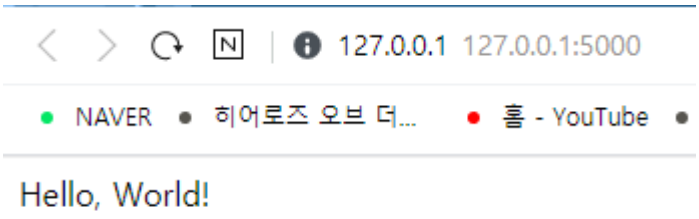
  주로 사용하는 인자는 다음과 같다.
  port (ex: port = 5000)

  수동적 특정 포트 번호를 잡고 싶다면 이걸 사용하자.
  host(ex: 0.0.0.0, 127.0.0.1, ...)

  해당 웹 서비스로 접근을 허용하는 IP 혹은 Domain을 적을 수 있다.
  예를들어 (127.0.0.1)를 보내면
  웹서버를 실행시킨 해당 PC 외에는 접속이 불가능 하지만,
  모든 IP를 뜻하는 (0.0.0.0)을 보내면 외부에서도 접속이 가능해지는 셈이다.

아래 명령어를 실행해 서버를 구동 시킨다.

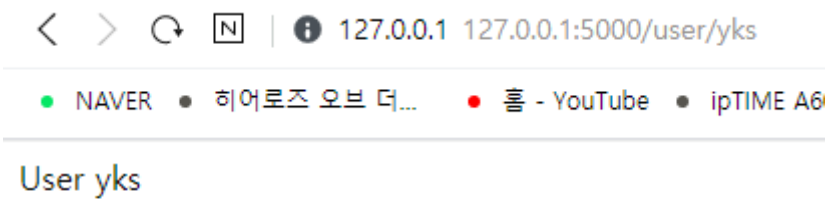
```
>python routes.py

 * Serving Flask app "routes" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production d
eployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



브라우저에서 전달

```
routes.py

from flask import Flask
app = Flask(__name__)

@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % username

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

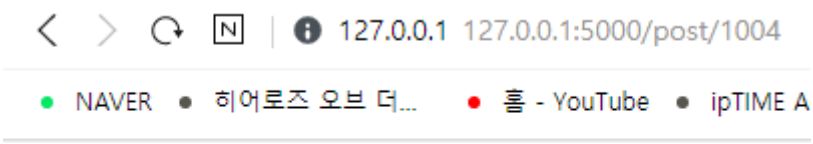
실행 결과 화면



숫자 전달받기

```
routes.py

from flask import Flask
app = Flask(__name__)

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id


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

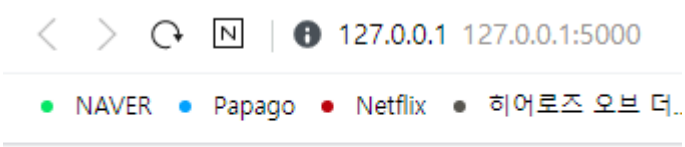
실행 결과 화면



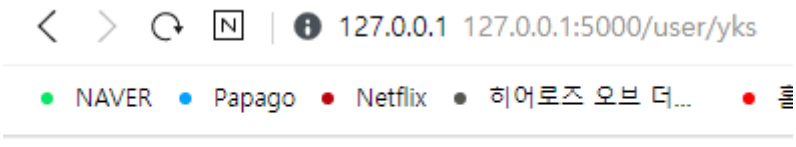
Post 1004

하나로 통합

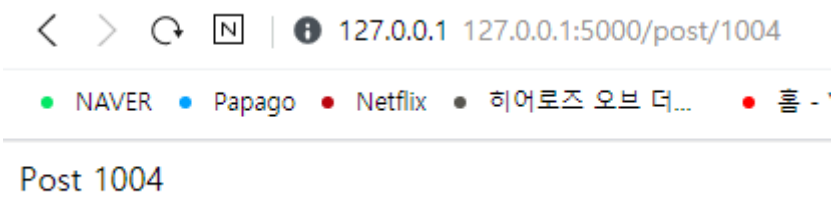
```
routes.py

from flask import Flask
app = Flask(__name__)

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

@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id


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



Hello, World!



User yks



인자로 받을 변수 규칙에는 string(%s), int(%d), float(%f) 등이 있으며,
아무것도 적지 않을 경우, 자동으로 string 형태로 인자를 주게 된다

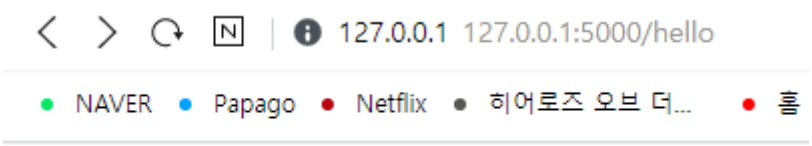
또한, 특정 함수 1개에 복수의 데코레이터를 덮어 씌울 수 있다.

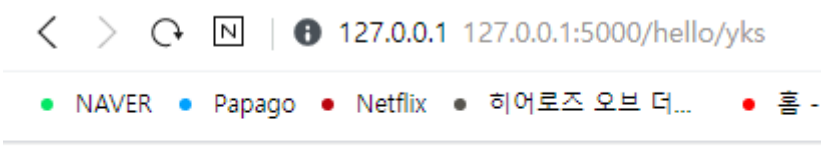
예를 들어 아래처럼 2개의 데코레이터를 씌워준 후,
기본 함수 인자 설정을 None 으로 해주면
인자가 없을 경우 첫번째 URL이고,
있을 경우 2번째 URL이 되는 것이다.

복수의 데코레이터

routes.py

```
from flask import Flask
app = Flask(__name__)

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

@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id

# 복수의 데코레이터
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return 'User %s' % name


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





템플릿 사용하기

이전까지는 그저 특정 문자열을 반환하는 것으로 웹 페이지를 보여주었지만 실제 웹 사이트들은 대부분 그렇지 않다.
사진이나 영상 같은 정적 파일 등과 예쁜 폰트 등의 글씨를 보여주기 위해 수많은 코드가 포함된 HTML 파일을 사용하는 것이 대부분이다.

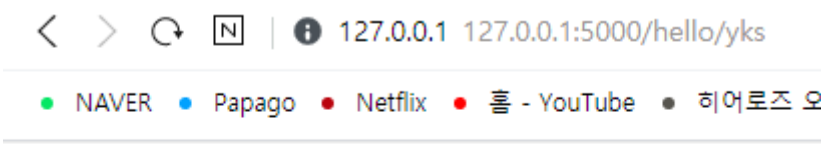
플라스크의 경우,
HTML 문서를 보관하기 위한 디렉터리를 따로 만들어 주어야 해당 파일들을 인식할 수 있는데,
해당 폴더의 위치는 반드시 플라스크를 실행시키는 코드와 같은 경로이며, templates라는 이름이어야 한다.

```
routes.py

from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<user>')
def hello_name(user):
    return render_template('hello.html', name = user)
    # render_template(): 외부파일을 불러낼때 필요한 함수

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

```
templates/hello.html

<!DOCTYPE html>
<html>
<body>

    <h1>Hello {{ name }}!</h1>
```

```
</body>
</html>
```

> `render_template()` : 플라스크에서 외부의 html파일을 불러오기 위해 사용하는 메소드.
> `templates` 폴더에서 인자로 받은 hello.html 파일을 탐색한 후,
> `name` 인자를 건네주며 페이지를 실행 시켰다.



Hello yks!

이렇게 { } 등을 사용하여
html 문서 내에서 직접 코드를 실행시키는 문법을 Template Engine이라고 부르며
이 템플릿 엔진의 이름은 Jinja2 엔진이다.

Jinja2

이 템플릿 엔진을 사용하여
HTML문서에서 몇 가지 파이썬 구문을 작성하고 실행시킬 수 있다.

코드에 용도에 따라 적절한 괄호를 감싸주어서 사용하면 된다.

- {{ … }}
  변수나 표현식의 결과를 출력하는 구분자(delimeter)
- {% … %}
  if문이나 for문 같은 제어문을 할당하는 구분자(delimeter)
- {# … #} : 주석
- {%- … %}, {%+ … %}, {% … -%} … : 공백 제거 혹은 유지 (trim 기능)
  기본적으로 jinja 템플릿 엔진은 템플릿 줄 끝의 ` ` 개행문자를 제외한
  나머지 공백은 그대로 출력
  실제로 보이는 화면에는 이러한 공백을 제외하고 싶을 경우에
  템플릿 태그 앞뒤에 +나 -를 붙여
  +를 붙으면 공백 유지, - 를 붙이면 공백 제거의 형태가 되는 것이다.
- {% raw %} … {% endraw %} : 이스케이프(escape)
  우리가 템플릿 형태와 동일한 “텍스트”를 출력하고자 할 때 사용.
  {% raw %} 와 {% endraw %} 사이에 이스케이프 할 문자들을 입력.

  그러나 {{ ‘{{’ }} 와 같이 따옴표로 간단하게 이스케이프 할 수도 있다.
  작은 문자열의 경우는 이런식으로 사용하는 것이 편리.

```
Jinja2 예제 코드
```

```
{% for <개별요소> in <리스트> %}
<실행코드>
{% endfor %}

{% for item in navigation %}
<li><a href="{{ item.href }}">{{ item.caption }}</a></li>
{% endfor %}

{% for n in datalist %}
<li>n</li>
{% endfor %}
```

```
{% if <조건> %}
    <실행코드>
{% elif <조건> %}
    <실행코드>
{% else %}
    <실행코드>
{% endif %}

{% <실행코드> if <조건> else <거짓말일때 실행코드 %}
{% for users in users %}
    <li>{{ user.username }}</li>
{% endfor %}
```

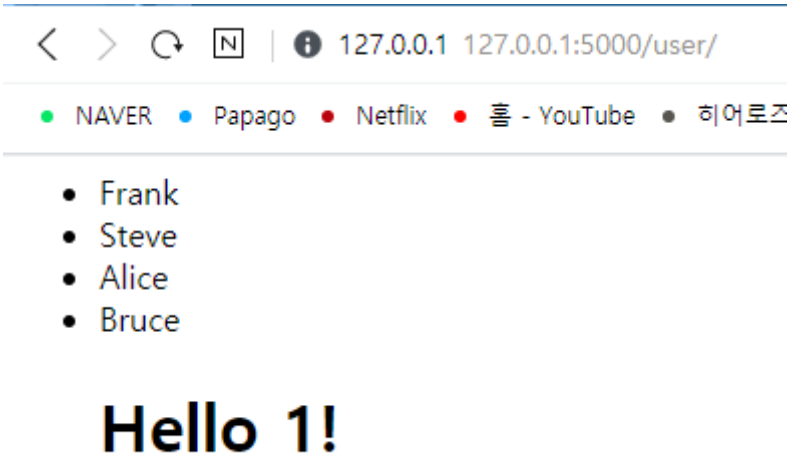
locals() 메소드

웹 페이지의 성격에 따라
HTML 문서에서 보여주어야 할 변수나 자료형이 매우 많을 수도 있다.

그래서 해당 함수 내의 지역 변수 전체를 인자로 보내주는 locals() 메소드를 사용하면 매우 편리하다.

```
routes.py

from flask import Flask, render_template
app = Flask(__name__)

@app.route('/user/')
def hello():

    users = ["Frank", "Steve", "Alice", "Bruce"]
    var = 1
    return render_template('user.html', **locals())

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

```
templates/user.html

<ul>
{% for user in users %}
    <li>{{ user }}</a></li>
{% endfor %}

{% if var %}
    <h1>Hello {{ var }}!</h1>
{% endif %}
</ul>
```



URL 및 HTTP 메소드 다루기

각 URL간의 이동 및 HTTP 메소드 사용법에 대해 간단히 알아보자

Redirect(리다이렉트)

Flask 에서는 URL 간의 쉽고 편한 이동을 위해 redirect 메소드를 제공한다.
사용방법은 아래와 같다.

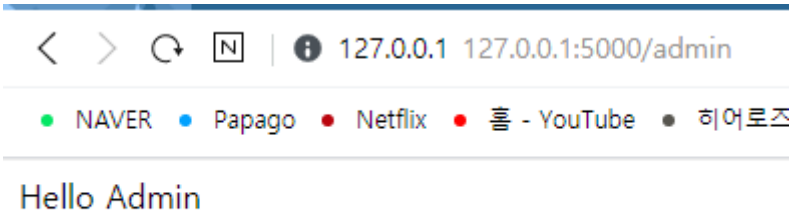
```
routes.py

from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
    return 'Hello Admin'

@app.route('/guest/<guest>')
```

```
def hello_guest(guest):
    return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
    if name == 'admin':
        return redirect('/admin')
    else:
        return redirect(url_for('hello_guest', guest = name))

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

`/user/<name>` URL에 입력된 name 값에 따라 다른 URL로 향하게 되어있다.
name 값이 admin 일 경우, /admin URL(hello_admin()) 을 실행시키고,
나머지는 `guest/<guest> URL(hello_guest)` 를 실행시키는 구조이다.

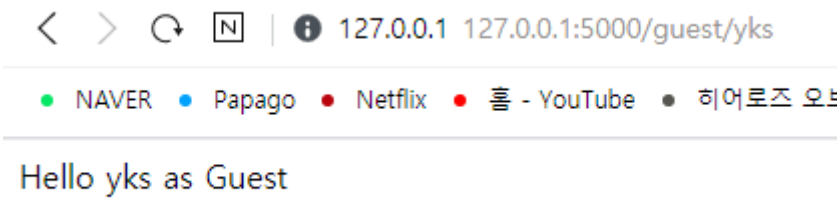
return redirect('/admin') == return redirect(url_for("hello_")) 해당 URL로 리다이렉트시키는 함수이다. URL을 줘도 되고, url_for() 메소드를 사용하여 실제 함수 이름을 인자로 주어도 된다.

```
http://127.0.0.1:5000/user/admin
```



> /admin 으로 이동된다.

```
http://127.0.0.1:5000/user/yks
```



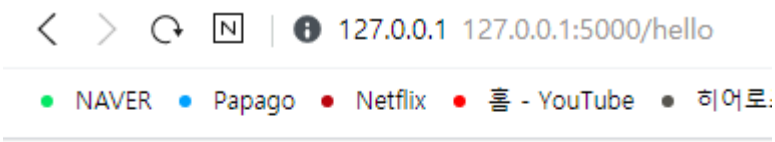
> 매개변수를 가지고 /guest 로 이동된다.

다음은 http 메소드를 통해 작동하는 웹 페이지를 만들어 보자.
http 프로토콜은 WWW(월드 와이드 웹)에서 데이터 통신의 기본 토대이다.

Flask를 사용해 GET 및 POST 메소드를 사용하는 코드를 만들어본다.

```
routes.py

from flask import Flask, redirect, url_for, request, render_template
app = Flask(__name__)

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

@app.route('/success/<name>')
def success(name):
    return 'welcome %s' % name

@app.route('/login', methods = ['POST', 'GET'])
# methods = ['POST', 'GET'] : 전송방식을 'POST', 'GET' 모두 사용하겠다는 의
미
def login():
    if request.method == 'POST':
        user = request.form['nm']
        return redirect(url_for('success', name=user))
    else:
        user = request.args.get('nm')
        return redirect(url_for('success', name=user))

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

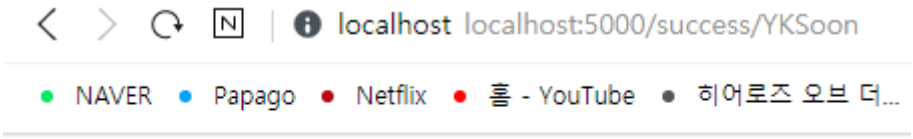
```
templates/login.html

<!DOCTYPE html>
<html>
<body>
    <form action="http://localhost:5000/login" method="POST">
        <p>Enter Name:</p>
        <p><input type="text" name="nm" /></p>
        <p><input type="submit" value="submit" /></p>
    </form>
</body>
</html>
```



Enter Name:

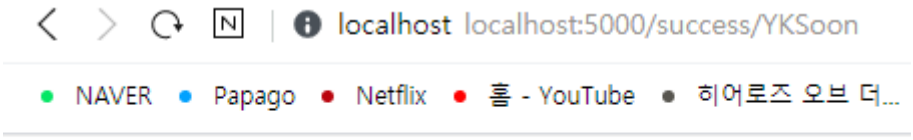
YKSoon

submit



welcome YKSoon

아래는 GET 방식이다



welcome YKSoon