Flask

Flask

- Flask
  - ✓ Python 기반 Micro Framework를 표방
  - ✓ 확장성 있는 설계를 지원
    - Django는 현재 가장 많은 사용자를 보유한 Python Web Application Framework 이며 강력하고 풍부한 기능을 제공하기 때문에 Framework 사용법만 잘 익힌다면 아주 사용하기 편리하지만 강력한 기능을 제공해 주는 대신 Framework 자체적으로 설계한 개발 패턴에서 크게 벗어날 수 없는 구조
    - Flask는 처음부터 주어진 기능이 없지만 내가 원하는 설계 방향대로 framework를 구축해 나갈 수 있으며 기능 보강을 위해서 확장 모듈을 사용
    - WSGI(Web Server Gateway Interface – 웹 서버와 웹 애플리케이션의 인터페이스를 위한 Python 프레임워크 용 Library인 Werkzeug(http://werkzeug.pocoo.org/) 와 HTML에 데이터를 렌더링하는 엔진인 Jinja2 template(http://jinja.pocoo.org/)을 사용
  - ✓ Linked In 사이트가 Flask를 사용하는 대표적인 서비스이고 Airflow 도 Web Server를 Flask를 이용
  - ✓ docs-kr: http://flask-docs-kr.readthedocs.io/ko/latest/index.html

Flask

- 프로젝트 생성 및 실행
  - ✓ 프로젝트를 위한 디렉토리 생성
  - ✓ 가상 환경 생성
  - ✓ Flask 설치: pip install flask
  - ✓ 디렉토리에 Python 파일을 생성하고 작성 – app.py

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

#시작 요청이 왔을 때 아래 코드를 수행
@app.route('/')
def main():
    return 'Hello Flask!'

#서버 구동
#app.run(host='0.0.0.0')
```

Flask

- 프로젝트 생성 및 실행
  - ✓ app.py
    - app = Flask(__name__)은 플라스크 애플리케이션을 생성하는 코드이며 이 코드에서 __name__이라는 변수에 모듈 이름이 담기는데 이 파일이 실행되면 app.py라는 모듈이 실행되는 것이므로 __name__ 변수에는 app 이라는 문자열이 담김
    - @app.route는 특정 주소에 접속하면 바로 다음 줄에 있는 함수를 호출하는 Flask의 데코레이터로 / 요청이 오면 아래 함수를 실행해서 브라우저에 출력
    - app.run() – 파일을 바로 시작할 때 작성
      - run 함수를 통해서 앱을 실행시키는데 기본적으로 localhost:5000에 열림
      - # host, port, threaded, debug 등의 매개변수가 있음
      - app.run(host='127.0.0.1', port=9055, threaded=True, debug=True)
      - 배포 단계가 아니라면 debug 매개변수를 True로 주는 게 좋음
      - 코드가 바뀌면 재시작

Flask

- ❖ 프로젝트 생성 및 실행
  - ✓ 실행
    - ● flask run 이라는 명령으로 서버를 구동

      flask run --host=0.0.0.0
      - ◆ app.py 파일을 실행하며 포트는 5000번이 기본
    - ● 시작 파일의 이름이 app.py 가 아닌 경우는 환경 변수의 값을 수정해야 함(export 명령 대신에 set 명령이나 $env= 으로 설정해야 하는 경우도 있음)

      export FLASK_APP=main.py
    - ● 개발 환경으로 실행하기 위한 환경 변수 설정

      export FLASK_ENV=development

Flask

- 요청 URL 처리
  - ✓ 여러 개의 요청을 처리하고자 하는 경우는 app.rote 라는 데코레이터에 매개변수로 요청 URL을 작성하고 함수를 만든 후 출력할 내용을 작성

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

@app.route('/')
def index():
    return '시작 페이지'

@app.route('/sports')
def hello():
    return '스포츠 페이지'

app.run(host='0.0.0.0')
```

Flask

- 요청 URL 처리
  - ✓ 요청 URL의 일부를 클라이언트에서 전송한 데이터로 사용
    - 변수로 사용할 부분에 <변수명>으로 설정
    - 앞에 int: float(자료형)를 붙이면 형 변환을 수행
    - 형 변환을 수행하다가 에러가 발생하면 400번대 에러 코드를 전송
    - URL을 만들 때 마지막에 /를 추가하면 URL을 요청할 때 /를 삽입하지 않더라도 자동으로 추가해서 처리
    - 마지막에 /를 추가하지 않으면 /를 추가해서 요청을 하면 안됨

Flask

- 요청 URL 처리
  - ✓ 요청 URL의 일부를 클라이언트에서 전송한 데이터로 사용

```
@app.route('/login/<userid>')
def userdisp(userid):
    return 'id는 ' +  userid + '입니다.'

@app.route('/post/<int:su>')
def sudisp(su):
    return 'su는 ' + str(su) + '입니다.'
```

Flask

- 요청 URL 처리
  - ✓ 데코레이터 없이 url 매핑
    - app.add_url_rule('url', 'endpoint', 호출될 함수이름)
    - add_url_rule이라는 함수로 리소스를 추가하면 됩니다.
    - rule, endpoint, view_func 순서로 설정

```
def index_view():
    return 'index'

app.add_url_rule('/index', view_func=index_view)
```

Flask

- 템플릿 엔진
  - ✓ Flask는 <http://jinja.pocoo.org/2/>_ 를 템플릿 엔진으로 구성하여 자동으로 HTML 이스케이핑
  - ✓ 템플릿을 출력하기 위해서는 render_template() 함수를 이용
    - 템플릿의 이름과 템플릿에 보여줄 변수를 키워드 인자로 넘겨주면 됨
  - ✓ 보여질 페이지의 기본 위치는 서버 파일과 동일한 위치의 templates 디렉토리

Flask

- 템플릿 엔진
  - ✓ 라우팅 처리 코드

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

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

app.run(host='0.0.0.0', debug=True)
```

Flask

- 템플릿 엔진
  - ✓ 프로젝트에 templates 디렉토리를 생성하고 login.html 파일을 생성하고 작성한 후 실행

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>로그인 페이지</title>
</head>
<body>
    <h2>안녕하세요 반갑습니다.</h2>
    <h3>로그인 페이지 입니다.</h3>
</body>
</html>
```

Flask

- 파라미터 전송 방식 처리
  - ✓ 파라미터 전송 방식에 대한 처리는 methods 옵션을 이용해서 설정
  - ✓ 요청을 처리하는 함수 내에서는 request.method 속성을 이용해서 처리
  - ✓ 대소문자 구분은 하지 않기 때문에 GET 이나 POST를 소문자로 입력해도 됨

Flask

- 파라미터 전송 방식 처리
  - ✓ URL 처리 함수 추가

```
@app.route('/param/', methods=['GET', 'POST'])
def param():
    if request.method == 'POST':
        return 'post 요청'
    else:
        return 'get 요청'
```

Flask

- 파라미터 전송 방식 처리
  - ✓ login.html 페이지를 수정한 후 실행

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>샘플 </title>
</head>
<body>
    <form action="http://localhost/param/" method="post">
        <input type="submit" value="post전송"/>
    </form>

     <form action="http://localhost/param/" method="get">
        <input type="submit" value="get전송"/>
    </form>
</body>
</html>
```

Flask

- 클라이언트 정보 읽기
  - ✓ Client가 보내 준 정보는 flask 패키지의 request 객체를 이용해서 읽을 수 있는데 이 객체는 전역 객체라서 import 하면 프로그램 내의 모든 곳에서 사용할 수 있음
  - ✓ request 객체의 속성
    - form: POST 요청으로 부터 넘어온 폼 데이터를 갖는 MultiDict
      - 파일을 전송한 경우는 이 속성이 아닌 files 속성에서 처리
    - args: url에 넘어온 파라미터의 내용을 갖는 MultiDict
    - values: form과 args의 내용을 갖는 CombinedMultiDict(query-string, form)
    - cookies: 요청과 같이 전송되는 쿠키의 내용을 갖는 dict
    - headers: dict 객체로 된 유입 요청 헤더
    - files: POST 요청으로 업로드 한 파일 정보를 갖는 MultiDict
      - 파일은 FileStorage 객체로 저장
      - Python에서 사용하는 표준 파일 객체처럼 동작하는데 차이점은 이 객체는 파일 시스템에 파일을 저장할 수 있는 save() 함수가 있음
    - method: 현재 요청에 대한 HTTP 함수(POST, GET etc.)

Flask

- 클라이언트 정보 읽기
  - ✓ request 객체의 속성
    - path
    - script_root
    - url
    - base_url
    - url_root
    - 사용자가 아래와 같은 URL을 요청한 경우

      http://www.example.com/myapplication/page.html?x=y

      - path: /page.html
      - script_root: /myapplication
      - base_url: http://www.example.com/myapplication/page.html
      - url: http://www.example.com/myapplication/page.html?x=y
      - url_root: http://www.example.com/myapplication/

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ FORM 데이터 읽기
    - login.html 페이지 수정

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>로그인 페이지</title>
</head>
```

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ FORM 데이터 읽기
    - ● login.html 페이지 수정

```
<body>
    <form method="post">
            <table border=0 width=400 height=100>
                <tr bgcolor="yellow">
                    <td align=right><font size=2>아이
디 :</font></td>
                    <td><input type="text" name="id"
size=10></td>
                </tr>
                <tr bgcolor="yellow">
                    <td align=right><font size=2>비밀번
호 :</font></td>
                    <td><input type="password"
name="pass" size=12></td>
                </tr>
```

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ FORM 데이터 읽기
    - login.html 페이지 수정

```
<tr bgcolor="yellow">
    <td align=right><font size=2>취미 :</font></td>
                                        <td>
        <input type="checkbox" name='hobby' value="reading"
style="width:60pt" size="10" />독서
        <input type="checkbox" name='hobby' value="programming"
style="width: 60pt" size="10" />프로그래밍
        <input type="checkbox" name='hobby' value="game"
style="width: 60pt" size="10" />게임
    </td>
                                    </tr>
<tr bgcolor="yellow">
                                        <td colspan=2 align=center>
                    <input type="submit" value="로그인" style="width:60pt"
size="10">
                    <input type="reset" value="다시 작성" style="width: 60pt"
size="10"></td>
                                    </tr>
                </table>
        </form>
```

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ FORM 데이터 읽기
    - login.html 페이지 수정

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

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ FORM 데이터 읽기
    - login 처리 메서드 수정

```
@app.route('/login/', methods=['POST', 'GET'])
def login():
    if request.method == 'POST':
        return request.form["id"] + ":" + request.form["pass"] + ":" + '
'.join(request.form.getlist("hobby"))
    else:
        return render_template('login.html')
```

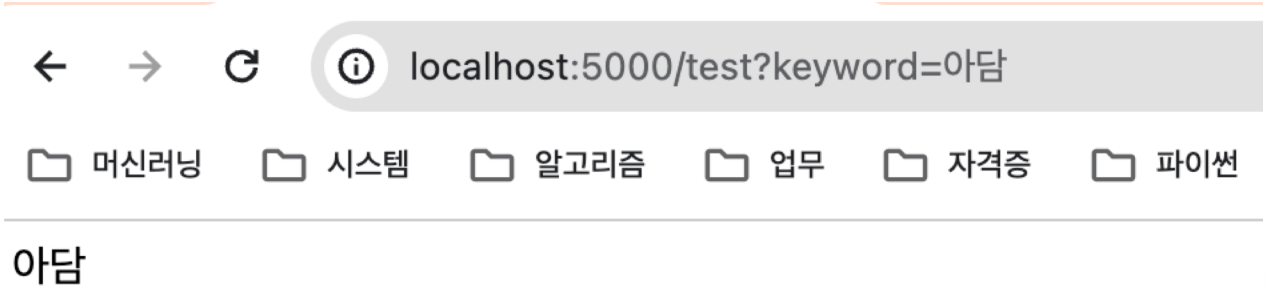
Flask

- 클라이언트 정보 읽기
  - ✓ Query String 읽기
    - test 요청 처리 메서드 추가

```
@app.route('/test', methods=['POST', 'GET'])
def test():
    return request.args["keyword"]
```

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ Query String 읽기
    - Postman 이나 브라우저에서 확인



Flask

- 클라이언트 정보 읽기
  - ✓ 파일 업로드 처리
    - form에 enctype="multipart/form-data" 를 설정하면 브라우저에서 파일을 업로드 할 수 있음
    - 업로드 된 파일들은 메모리나 파일 시스템의 임시 장소에 저장
    - files 객체의 files 속성을 찾아 그 파일들에 접근할 수 있으며 업로드 된 각 파일들은 dictionary 안에 저장되어 있음
    - 표준 Python file 객체처럼 사용할 수 있으며 서버의 파일 시스템에 파일을 저장하도록 하는 save() 함수를 가지고 있음
    - 애플리케이션에 파일이 업로드 되기 전 클라이언트에서의 파일명을 알고 싶다면 filename 속성에 접근할 수 있음

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ 파일 업로드 처리
    - ● upload.html 페이지 생성

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>파일 업로드</title>
</head>
<body>
    <form action="./" method="post" enctype="multipart/form-data">
                    문자열: <input type="text" name="text" /> <br />
                    파일: <input type="file" name="file" /> <br />
                    <input type="submit" value="전송" />
        </form>
</body>
</html>
```

Flask

- ❖ 클라이언트 정보 읽기
  - ✓ 파일 업로드 처리
    - upload 요청을 처리하는 코드를 작성

```
@app.route('/upload/', methods=['POST', 'GET'])
def upload():
    if request.method == 'POST':
        f = request.files['file']
        f.save(f.filename)
        print(request.form['text'])
        return '파일 업로드 성공'
    else:
        return render_template('upload.html')
```

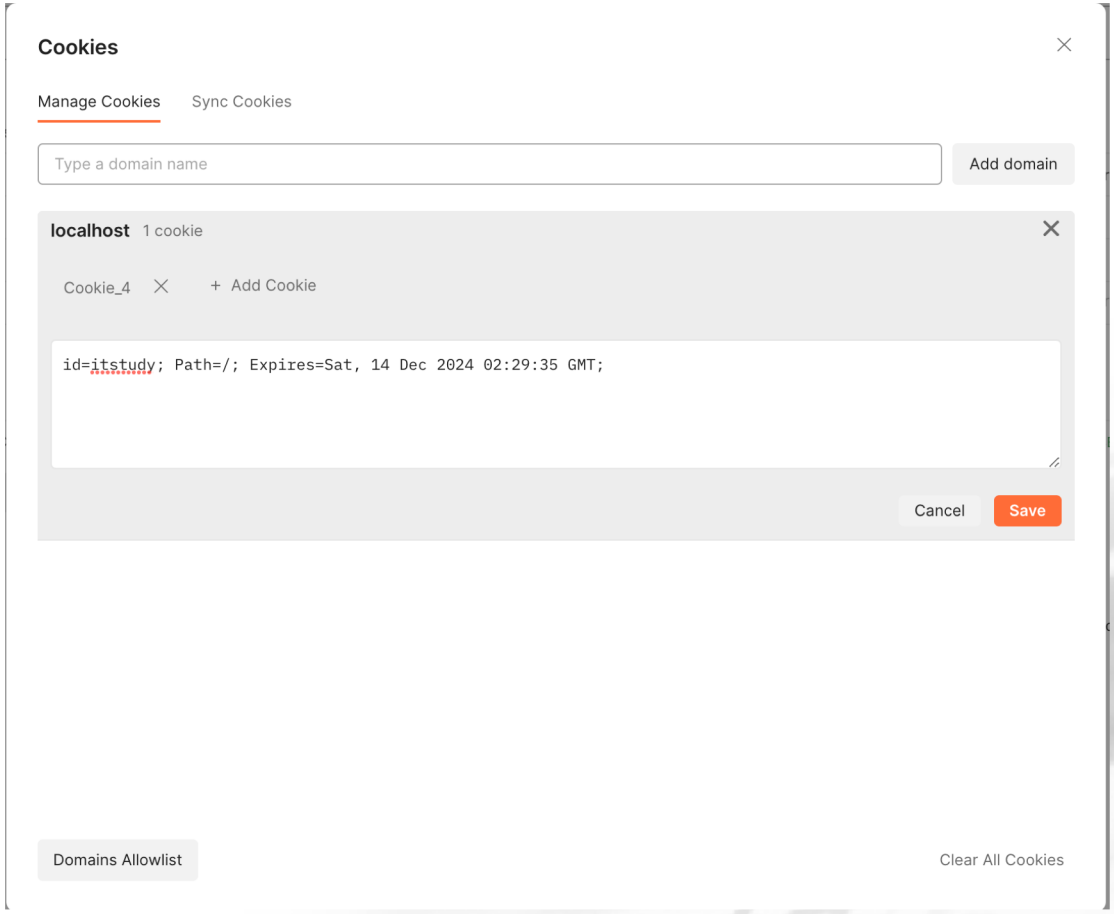
Flask

- ❖ 클라이언트 정보 읽기
  - ✓ Header 와 Cookie 정보 읽기
    - ● cookie 와 header 요청을 처리하는 코드를 작성

```
@app.route('/cookieheader/', methods=['GET', 'POST'])
def cookieheader():
    id = request.cookies.get('id')
    userAgent = request.headers.get('User-Agent')
    name = request.headers.get('Name')
    return "cookie:" +  id + " userAgent:" + userAgent + " name:" + name
```

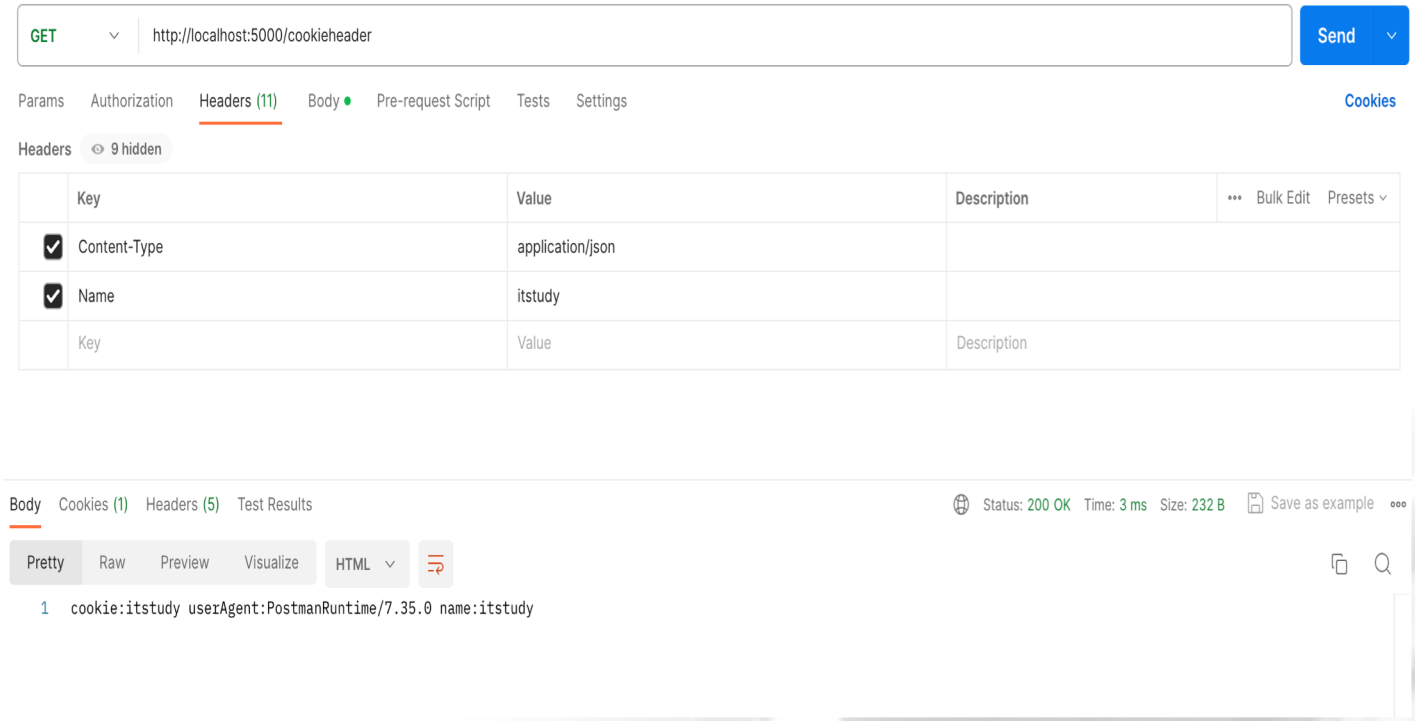
Flask

- ❖ 클라이언트 정보 읽기
  - ✓ Header 와 Cookie 정보 읽기
    - ● 확인



Flask

- ❖ 클라이언트 정보 읽기
  - ✓ Header 와 Cookie 정보 읽기
    - 확인



Flask

- Rest API
  - ✓ JSON 출력
    - flask의 jsonify는 flask.wrappers.Response의 객체를 반환
    - Content Type이 application/json으로 설정되고 반환 데이터는 JSON 문자열로 처리

```
from flask import Flask, request
from flask import render_template
from flask import jsonify

app = Flask(__name__)
@app.route('/json/')
def json():
    response = [{'name': 'Park', 'age': 18}, {'name': 'Kim', 'age': 18}]
    return jsonify(response)

app.run(host='0.0.0.0', debug=True)
```

Flask

- Rest API
  - ✓ XML 출력을 위해서는 결과를 return 할 때 (xml 문자열, {'Content-Type': 'text/xml'}) 의 형태로 리턴

```
@app.route('/xml/')
def ajax_ddl():
    result = '<persons>'
    result += '<person><name>Park</name><age>18</age></person>'
    result += '<person><name>Kim</name><age>18</age></person>'
    result += '</persons>'
    return result, {'Content-Type': 'text/xml'}
```