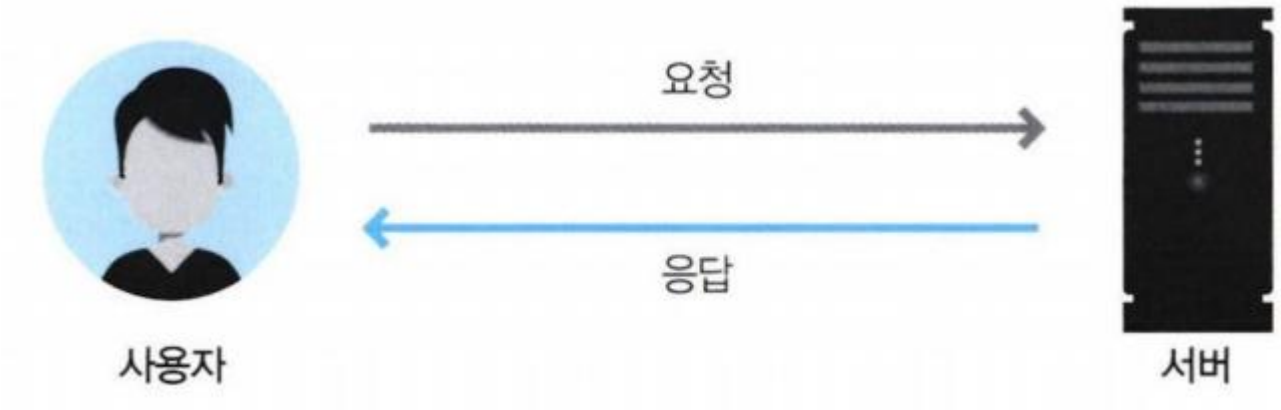
Flask

웹 프로그램 통신

웹의 통신방법

■ HTTP 통신



웹프로그램 통신

- 모든 웹프로그램은 사용자가 웹 브라우저를 이용해서 웹프로그램이 가지고 있는 자원(상품 정보, 강좌 목록(동영상))을 요청하면 웹브라우저가 이해할 수있는 형태로 재가공하거나, 있는 자원 그대로 웹브라우저에게 반환해준다.

- 웹서버와 웹브라우저 간에 발생하는 자원 반환 단계에서는 웹서버가 콘텐츠 협상(contents negitiation) 이라는 단계를 거친 후에 웹브라우저에 결과를 반환한다.

- 이 단계는 웹브라우저가 무엇을 처리할 수 있는지 웹서버와 협상하는 단계이다.

웹프로그램 통신

- 협상단계는 3가지로 분류할 수 있다.

1. 서버 기반의 협상 : 웹서버가 웹브라우저에 반환할 데이터의 형태를 직접 결정

2. 에이전트 기반의 협상 : 캐시서버 기반의 협상으로 웹 서버가 응답할 데이터 처리 형태를 결정하기 위해 첫 번째 수신을 처리한 에이전트에 의해 형태를 결정

3. 투명한 협상(1,2를 혼합한 협상)

협상에 필요한 HTTP 메시지 헤더

- **Accept : 브라우저가 처리할 수 있는 데이터의 형태와 선호도 ex.text/html**
- **Accept-Language : 브라우저가 수용할 수 있는 응답결과의 언어와 선호도 ex. kr, en**
- **Accept_Encoding : 브라우저가 수용할 수 있는 응답 인코딩 형태와 선호도 ex. gzip, br**

> HTTP 메시지는 요청(Request)메시지와 응답(Response)메시지로 나눈다.

- **HTTP 요청 메시지 : 메서드(GET, POST), HTTP버전(HTTP/1.1), 호스트명, 웹브라우저가 무엇인지 (User_Agent헤더), 어떤 언어(ko_KR, ko; en-US)/자원 형태 를 받아들이는지 기록**
- **HTTP 응답 메시지 : 첫행에는 HTTP 버전, HTTP 상태코드(200:성공), 상태코드 문자열 OK**

  두번째 행부터는 순서 없이 정보를 기술한다.

  날짜, 서버의 종류, 사용자 정의 헤더도 포함된다. 'X-'시작하는 헤더명은 사용자 정의 헤더라고 본다.

  (사용자 정의 헤더는 웹프로그램과 웹브라우저가 해석할 수 있을 때 의미있는 정보가 된다.)

파이썬을 위한 웹프로그램 통신 규약

- 웹프로그램은 사용자가 보낸 요청과 요청을 처리한 결과를 웹서버를 경유해서 주고받는다.
- 이 때 웹서버와 웹프로그램간의 메시지를 주고 받기 위한 약속이 필요한데 이 약속을 **CGI 규약**이라고 한다.
- **CGI(Common Gateway Interface) : CGI** 프로그래밍을 하기 위한 언어는 환경변수나 표준 입출력을 다룰 수 있는 언어라면 모두 사용 가능하지만, 실행 속도나 개발 편의성을 고려하여 **2000**년대 초까지는 대부분 **Perl** 언어를 사용하였다. 소스코드의 보안성을 위해 **C, C++,** 델파이와 같은 언어를 사용하는 경우도 있지만, 이 언어들은 웹프로그래밍에 특화된 언어가 아니어서 유지, 보수의 어려움이 있어 많이 사용되지는 못함.
- 파이썬은 **cgi** 모듈을 통해 **CGI** 환경변수와 **CGI** 표준 입출력에 직접 엑세스 해서 웹프로그램을 작성할 수 있다.
- 웹프로그램은 웹서버와는 독립적이어야 하는데 파이썬은 **WSGI** 표준을 통해 이 독립성을 구현한다.

  (WSGI 표준을 따르면 웹서버의 종류와는 상관없이 동작이 된다**)**
- **Flask**는 **Werkzeug**(벡자이그, 독일) 기반으로 작성되는데, 이 벡자이그가 **WSGI** 코어와 **URL** 라우팅을 지원하고 있다.

파이썬 웹 프레임워크

■ 마이크로 프레임워크 (Micro Framework)

: 웹 프로그래밍에 있어서 가장 핵심적인 요소만을 포함하고 있는 프레임워크

- 파이썬 : **Flask(WSGI 구현체인 Werkzeug와 템플릿 Jinja2), Bottle**
- 루비 : 시나트라(**Sinatra**) - 마이크로 프레임워크의 원조

■ 풀스택 프레임워크 (Full Stack Framework)

: 웹프로그래밍을 할 때 필요로 하는 모든 것들을 종합적으로 갖추고 있는 프레임워크

: 인증과 권한, **ORM**, 템플릿 라이브러리, 국제화와 지역화, 관리자, 보안 등의 여러요소를 갖춤

- 파이썬 : **Django, Web2py, Turbogear**
- 자바 : 스프링(**Spring**)
- 루비 : 레일즈(**Rails**)

Flask 주요 기능

Flask 주요 기능

주요기능	기능설명
Debug mode	개발 서버를 시작하는 것 이상의 작업을 수행할 수 있음 디버그 모드를 활성화하면 코드가 변경되면 서버가 자동으로 다시 로드되고 요청 중에 오류가 발생하면 브라우저에 대화형 디버거가 표시됨
Routing	@route 데코레이터를 사용해서 쉽게 함수를 URL에 바인딩 할 수 있음
Rendering Templates	Flask 프레임워크는 Jinja2 템플릿엔진을 기본으로 사용. jinja2 를 사용하면 render_template() 함수를 이용하여 HTML 렌더링 가능

Debug Mode

- **Debug Mode**

  - Debug mode를 활성화하면 코드가 변경되면 서버가 자동으로 다시 로드되고

  요청 중에 오류가 발생하면 브라우저에 대화형 디버거가 표시됨

```
(venv) seojh@linuxgeek:~/flask_project$ flask --debug run
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 131-363-590
```

Routing

- **Routing**

  - @route 데코레이터를 사용해서 URL을 쉽게 함수에 바인딩 가능

  - <text>와 같이 표시해 URL에 변수를 추가 가능

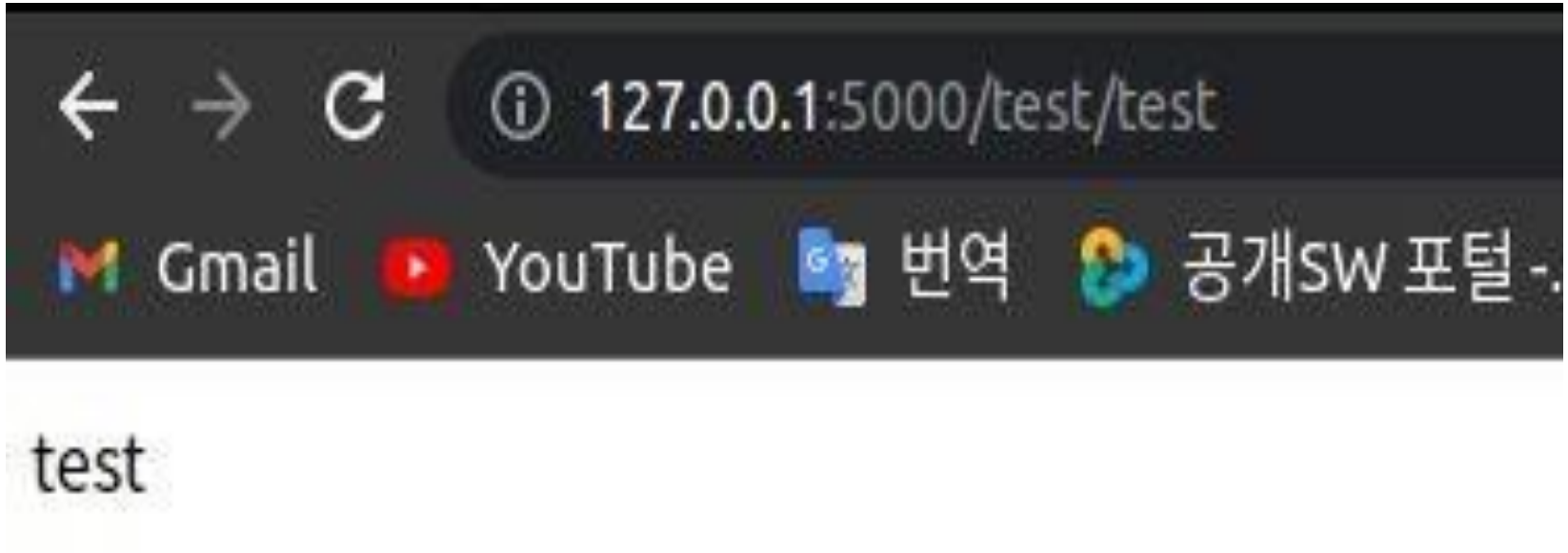
  - 그런 다음 text를 함수 인수로 받고 사용 가능

```
@app.route("/test/<text>")
def route_sample(text):
    return text
```

Routing

- **Routing**

  - 바인딩된 **url**로 접속한 결과 변수로 받은 **test**가 출력된 페이지



Template rendering

- templates rendering

- Jinja2 템플릿 엔진

- Flask 설치할 때 같이 설치되기 때문에 추가 설치 할 필요가 없다

- 플라스크의 템플릿 파일들은 기본적으로 /templates/ 폴더에 저장한다

- 웹 서버에서 페이지를 렌더링 해주기 위해서는 render_template()라는 함수를 임포트해서 사용하면 됨

app.py

```
1   from flask import Flask ,render_template
2
```

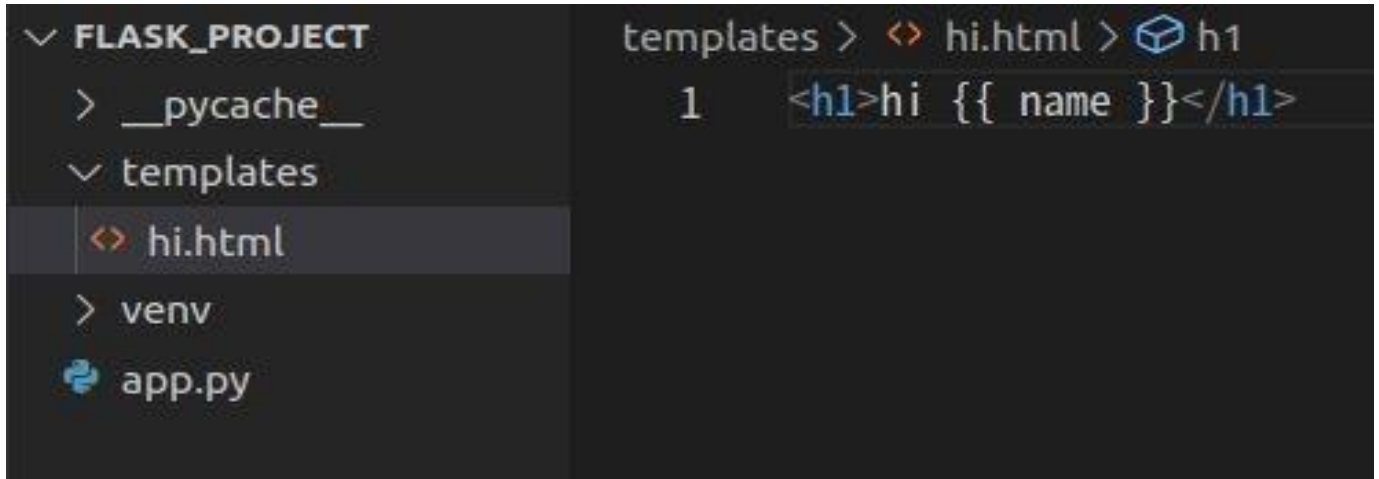
Template rendering

- templates rendering

- name변수를 템플릿으로 넘겨줌

```
@app.route("/hi/<name>")
def hi_template_render(name):
    return render_template('hi.html',name=name)
```

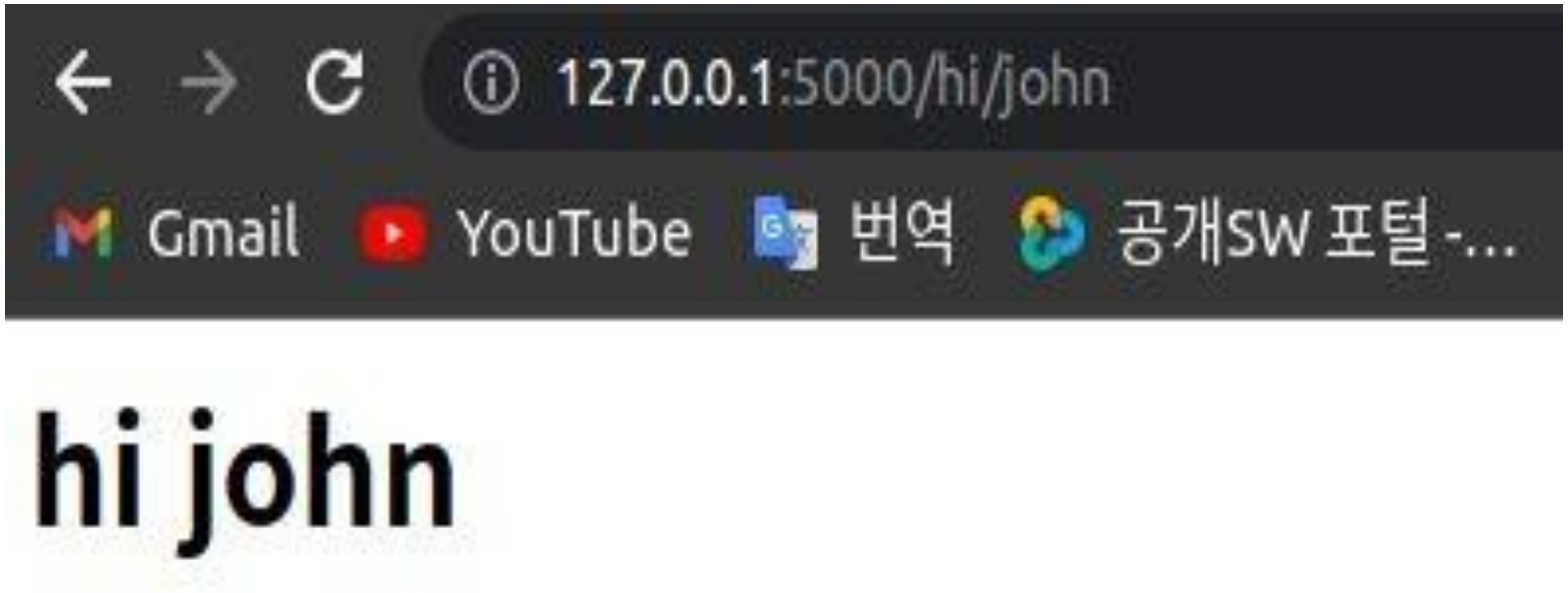
- Jinja2의 문법으로 name 변수를 사용



Template rendering

- templates rendering

- - '/hi/john' URL에 연결된 html 렌더링 결과 URL에 사용한 변수가 같이 출력된 것을 볼 수 있음



Jinja2 template 표현식

- {% : 템플릿에서의 프로그래밍 영역(제어문 등)을 넣기위해 시작하는 기호
- %} : 프로그래밍 영역을 기술을 끝내고 프로그래밍 영역을 종료하기 위해 사용하는 기호

- {{ : 변수를 출력하기 위해 시작하는 기호
- }} : 변수 출력이 끝나고 나서 사용하는 기호

- {# : 주석을 넣기 위해 시작하는 기호
- #} : 주석을 넣고 종료하기 위해 사용하는 기호

REST

Representational Status Transfer

- **REST**

자원을 이름으로 구분하여 해당 자원의 상태를 주고받는 모든 것

1. HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고,
2. HTTP Method(POST, GET, PUT, DELETE, PATCH 등)를 통해
3. 해당 자원(URI)에 대한 CRUD Operation을 적용하는 것을 의미

- **CRUD Operation**

- Create : 데이터 생성(POST)
- Read : 데이터 조회(GET)
- Update : 데이터 수정(PUT, PATCH)
- Delete : 데이터 삭제(DELETE)

Representational Status Transfer

- **REST 특징**

  자원을 이름으로 구분하여 해당 자원의 상태를 주고받는 모든 것

1. Server-Client(서버-클라이언트 구조)
2. Stateless(무상태)
3. Cacheable(캐시 처리 가능)
4. Layered System(계층화)
5. Uniform Interface(인터페이스 일관성)

Representational Status Transfer

REST 장점

- HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구출할 필요가 없다.
- HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해 준다.
- HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다. (멀티 플랫폼)
- Hypermedia API의 기본을 충실히 지키면서 범용성을 보장한다.
- REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악할 수 있다.
- 여러 가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.
- 서버와 클라이언트의 역할을 명확하게 분리한다. (생산성 향상)

REST 단점

- 표준이 자체가 존재하지 않아 정의가 필요하다.
- HTTP Method 형태가 제한적이다.
- 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.
- 구형 브라우저에서 호환이 되지 않아 지원해주지 못하는 동작이 많다.(익스폴로어)

<Request>

```
GET /api/product/2
```

<Response>

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
        "name": "보조배터리",
        "price": 10000,
        "description": "없음",
        "stock": 2
}
```

REST API

- **REST API**

  REST의 원리를 따라는 API

- **REST API 디자인 가이드**

  1) URI는 동사보다는 명사를 사용

```
GET /members/delete/1 (x)
GET /members/1 (o)
```

  2) URI 마지막에 '/'를 사용하지 않는다

```
http://example.com/member/1/ (X)
http://example.com/member/1   (0)
```

  3) 대문자 보다는 소문자를 사용

REST API

- **REST API 디자인 가이드**

4) 언더바(_)를 사용하지 않는다. URI가 길어질 경우 하이픈(-)을 사용한다

```
GET /example.com/test_blog (x)
GET /example.com/test-blog (o)
```

5) 파일 확장자는 URI에 포함하지 않는다

```
GET /example.com/photo.jpg (x)
GET /example.com/photo      (o)
```

6) 리소스 간의 관계를 표현할 때는 자원/ID/자원 형태를 사용한다

```
GET /users/1/product      (일반적으로 소유 'has' 관계 표현시)
```

RESTful

Restful

REST의 원리를 따르는 시스템

REST를 사용했다 하여 모두가 RESTful 한 것은 아닙니다.

REST API의 설계 규칙을 올바르게 지킨 시스템을 RESTful하다 말할 수 있습니다

Restful 목적

- 이해하기 쉽고 사용하기 쉬운 REST API를 만드는 것

- RESTful한 API를 구현하는 근본적인 목적이 성능 향상에 있는 것이 아니라
  일관적인 컨벤션을 통한 API의 이해도 및 호환성을 높이는 것이 주 동기이니,
  성능이 중요한 상황에서는 굳이 RESTful한 API를 구현할 필요는 없다.

- RESTful 하지 못한 경우
  ex1) CRUD 기능을 모두 POST로만 처리하는 API
  ex2) route에 resource, id 외의 정보가 들어가는 경우(/students/updateName)

쿠키와 세션

쿠키 (Cookie)

HTTP의 일종으로 사용자가 어떠한 웹 사이트를 방문할 경우,
그 사이트가 사용하고 있는 서버에서 사용자의 컴퓨터에 저장하는 작은 기록 정보 파일.
HTTP에서 클라이언트의 상태 정보를 클라이언트의 PC에 저장하였다가
필요시 정보를 참조하거나 재사용할 수 있다

- 쿠키 특징
  1. 이름, 값, 만료일(저장 기간 설정), 경로 정보로 구성되어 있다.
  2. 클라이언트에 총 300개의 쿠키를 저장할 수 있다.
  3. 하나의 도메인 당 20개의 쿠키를 가질 수 있다.
  4. 하나의 쿠키는 4KB(=4096byte)까지 저장 가능하다.

쿠키 (Cookie)

- 쿠키 동작 순서
  1. 클라이언트가 페이지를 요청 (사용자가 웹페이지 접근)
  2. 웹서버가 쿠키를 생성
  3. 생성한 쿠키에 정보를 담아 HTTP 화면을 응답할 때 같이 클라이언트에게 응답함.
  4. 넘겨받은 쿠키는 클라이언트가 가지고 있다가, 다시 서버에 요청할 때 요청과 함께 쿠키를 전송
  5. 동일 사이트 재방문시 클라이언트 PC에 해당 쿠키가 있는 경우, 요청 페이지와 함께 쿠키를 전송

쿠키 (Cookie)

- Flask – cookie
  쿠키는 기본적으로 쿠키 이름과 쿠키 값으로 구성

> 쿠키 특성
: 쿠키는 정해진 시간동안 유지된다.
: 쿠키는 지정된 웹사이트의 경로에 영향을 미친다.
: 쿠키는 지정된 도메인 주소에 영향을 미친다.

쿠키 특징

1. 이름, 값, 만료일(저장 기간 설정), 경로 정보로 구성되어 있다.
2. 클라이언트에 총 300개의 쿠키를 저장할 수 있다.
3. 하나의 도메인 당 20개의 쿠키를 가질 수 있다.
4. 하나의 쿠키는 4KB(=4096byte)까지 저장 가능하다.

쿠키 (Cookie)

- Flask – cookie
  쿠키는 기본적으로 쿠키 이름과 쿠키 값으로 구성

> set_cookie 인자

- key : 쿠키 이름(쿠키를 설정할 때 반드시 이름을 지정해야 한다.)
- value : 쿠키 값. 기본값은 빈 문자열
- max_age : 쿠키 지속시간. 기본값은 None인데 None이면 브라우저가 닫힐 때 자동으로 쿠키가 제거된다.
  시간 단위는 초단위의 시간값으로 전달한다.
  시간설정되면 초단위의 시간이 지나면 해당쿠키가 삭제된다.
- domain : 쿠키의 영향력이 미치는 도메인 주소. 기본값은 None이다.
- path : 쿠키의 영향력이 미치는 웹사이트의 경로. 기본값은 "/" 이다.

세션 (Session)

- Session

세션(session)이란 웹 사이트의 여러 페이지에 걸쳐 사용되는 사용자 정보를 저장하는 방법을 의미.
사용자가 브라우저를 닫아 서버와의 연결을 끝내는 시점까지를 세션이라고 합니다.
쿠키가 클라이언트에 저장되다보니 보안상 위험.
따라서, 서버에 데이터를 저장하는 방식인 세션이 많이 사용되는 추세. (권장)

> Flask 세션 관련 키

- SECRET_KEY : 비밀키
- SESSION_COOKIE_NAME : 세션 쿠키 이름, 기본값은 session
- SESSION_COOKIE_DOMAIN : 세션 쿠키가 동작할 도메인 주소, 설정하지 않았을 경우 SERVER_NAME에 있는 도메인에서 동작
- SESSION_COOKIE_PATH : 세션 쿠키가 동작할 URL 경로, 기본값은 /
- SESSION_COOKIE_HTTPONLY : 웹어플리케이션이 HTTP 프로토콜로 동작할 때만 세션 쿠키를 웹어플리케이션에 전송. 기본 설정값은 True
- SESSION_COOKIE_SECURE : 웹어플리케이션이 HTTPS프로토콜로 동작할 때만 세션 쿠키를 웹어플리케이션에 전송. 기본 설정값은 False
- # PERMANENT_SESSION_LIFETIME : 세션의 유효시간을 지정한다. 기본값은 31일

감사합니다