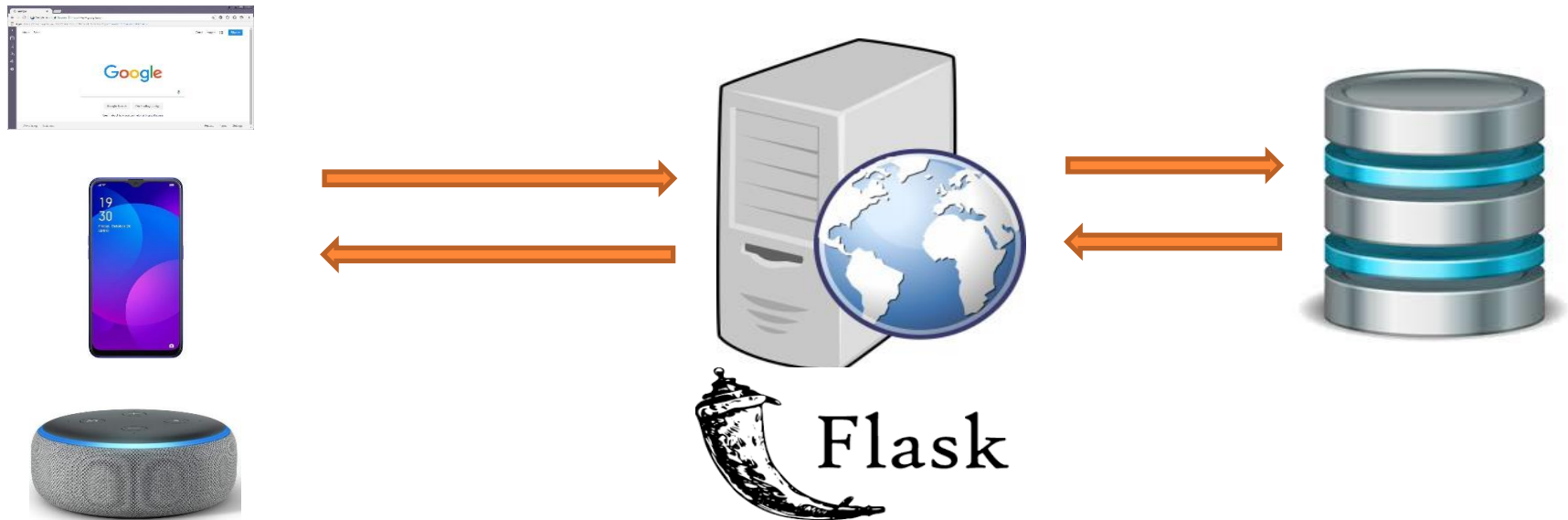


Flask Framework

김진숙

[Flask 프레임워크를 이용한 웹 개발]

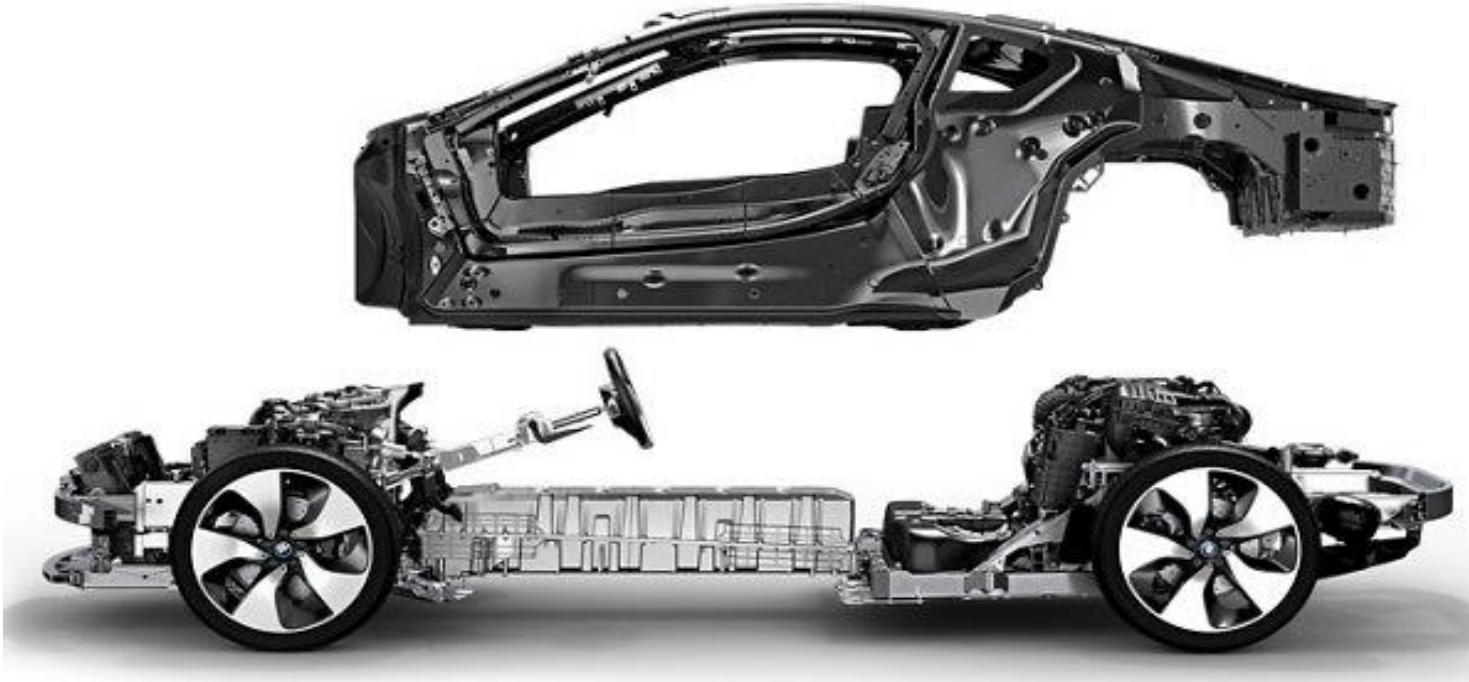
- 웹 프레임워크(Web Framework)
 - 웹 애플리케이션 프레임워크(Web Application Framework)
 - 웹 서비스 개발을 위한 프레임워크
 - Java의 Spring, Python의 Django



▪ Web Framework 종류

- Spring(Java)
- Django(Python)
- NodeJs(Javascript)
- **Flask**(Python)
 - Python 경량 웹 개발 프레임워크
 - Flask에서 HTTP API 제공
- Laravel(PHP)
- Rails(Ruby)

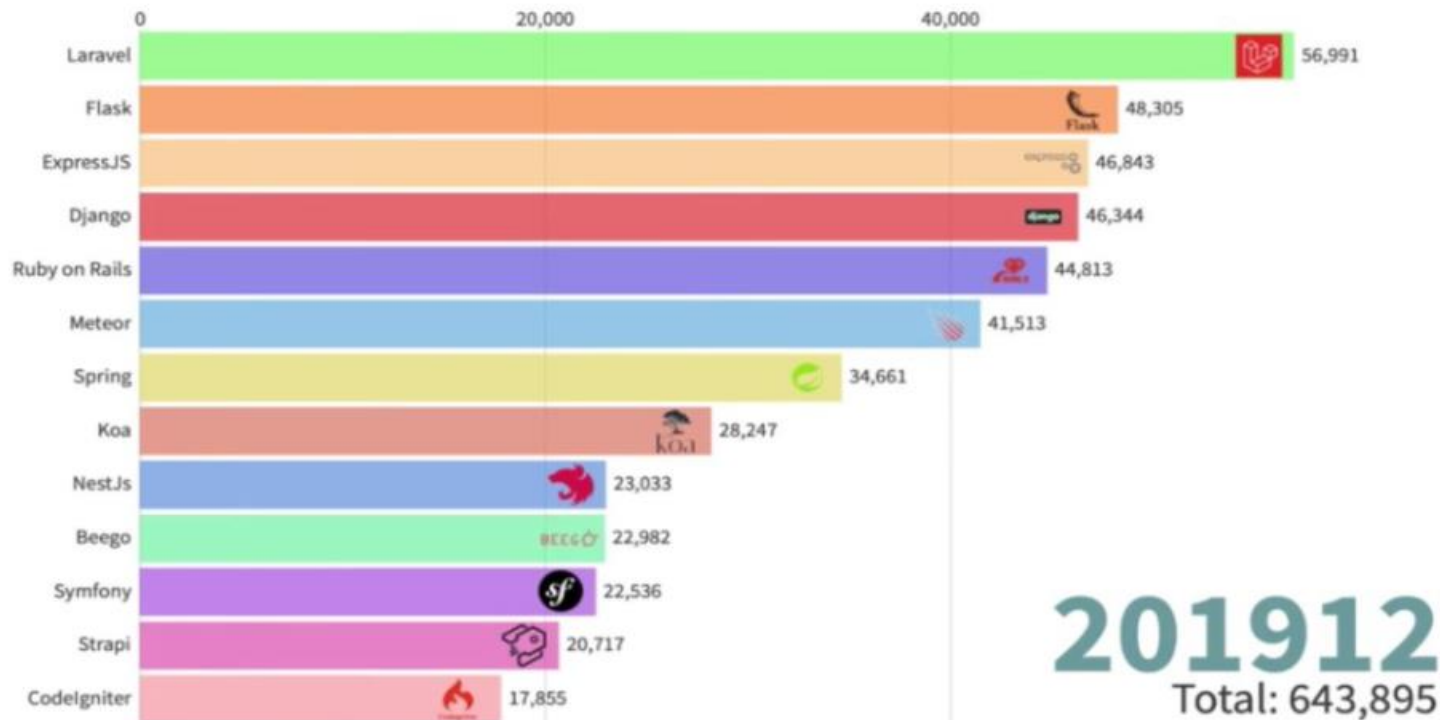
- Web Framework
 - 웹 개발을 위한 SW 반제품
 - 여러 기능들이 이미 개발 되어있고 Customizing을 할 수 있음
 - 보안, 응답/요청 처리, DB 연동 등의 공통 기능이 개발되어 있음



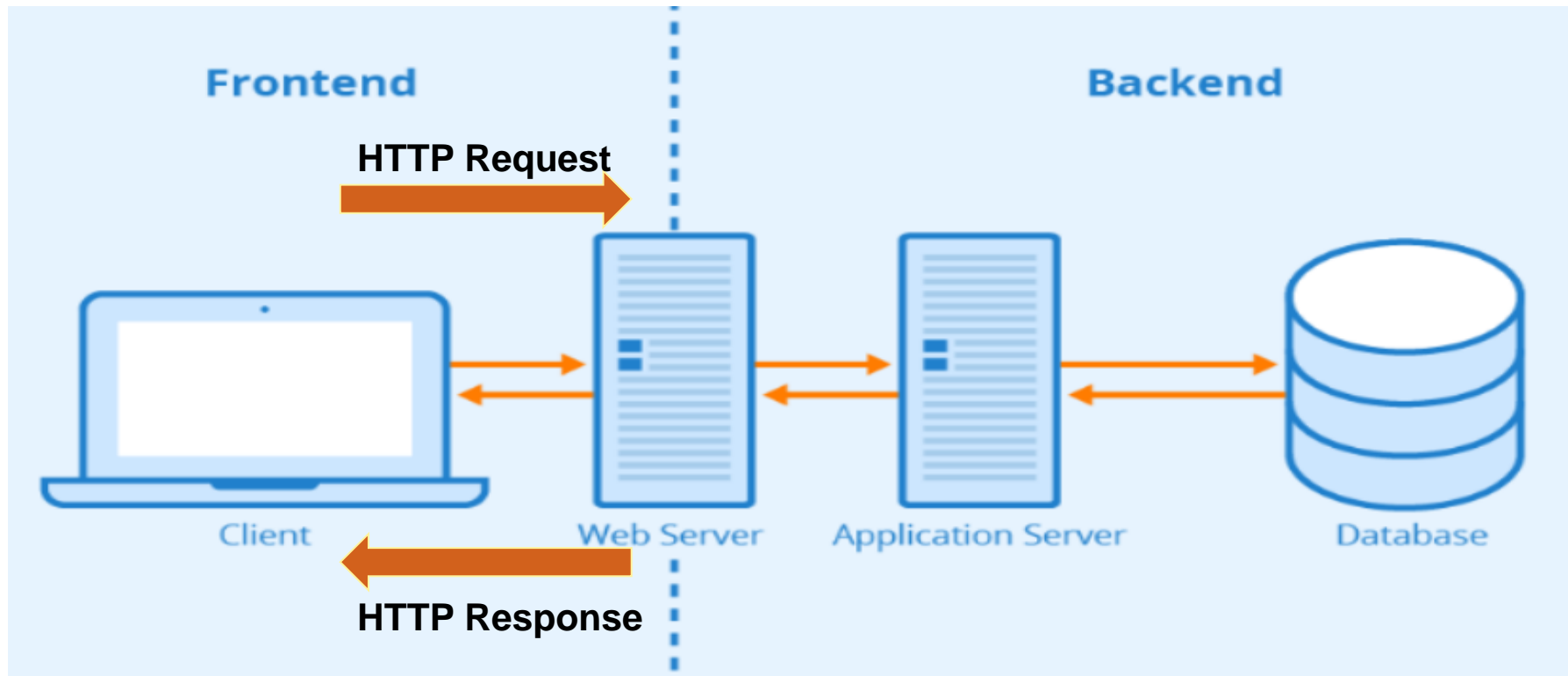
■ 플라스크(Flask)

- 파이썬을 사용하므로, 다양한 파이썬 라이브러리 활용 가능함(확장성이 매우 큼)
- Rest API, CORS, 데이터베이스 연결, MVC 등 가장 기본적인 기술을 다룰 수 있으며, 이는 다른 언어와의 프레임워크에 그대로 적용 가능
- 백엔드 이해와 구현 경험을 기반으로 JAVA, Spring, Go 등 다른 언어로 구현을 대체하면 빠르게 다양한 기술을 익힐 수 있음

Most popular backend frameworks 2012-2019 (based on Github stars)



- 프론트엔드(Frontend)와 백엔드(Backend)의 기본 구성



이미지 참조 : <https://blog.dalso.org/language/web/6523>

- **웹서비스 개발과 파이썬 flask 를 활용한 웹서비스 개발**
 - 1세대: USER ---- INTERNET --- WEB SERVER (Read static HTML)
 - 2세대: USER ---- INTERNET --- WEB SERVER (Create request-based HTML from CGI + DB)
 - 3세대: USER ---- INTERNET --- WEB SERVER MVC 패턴 기반 프레임워크 활용
 - MVC: Model - View - Control 패턴으로 구조화된 프레임워크를 사용, 빠르게 다양한 기능을 제공
 - 4세대: OpenAPI, RestAPI 를 혼합하여 다양한 서비스 제공, 다양한 웹 서비스 환경 개발
- **풀스택 프레임워크와 마이크로 프레임워크**
 - **풀스택 프레임워크**
 - 웹 개발에 관련된 모든 기능을 제공
 - JAVA Spring, Python Django, Ruby on Rails
 - 요청/응답 추상화, 세션 상태 관리, 사용자 인증/권한 관리, 웹페이지 템플릿, URL 매핑, 데이터베이스 접근, 보안, 캐시, 데이터 접근 추상화등 다양한 기능을 제공
 - **마이크로 프레임워크**
 - 웹 개발에 필요한 최소 기능만 제공, 나머지 기능은 자신이 원하는 다른 라이브러리나 프레임워크를 확장해 사용
 - Python Flask
 - 빠르게 원하는 기능을 기반으로 웹서비스 구축 가능, 이후 필요한 기능만 확장 가능

▪ Library와 Framework 비교

▪ Library

- 사용자가 library 코드를 호출
- 로직의 제어권이 사용자 코드에 존재

▪ Framework

- framework 코드가 사용자의 함수를 호출
- 전체 로직의 제어권을 framework에 존재
- 사용자는 framework이 규정 지은 규칙대로 코딩해야 함

- Flask 특징
 - 마이크로 프레임워크 기반
 - 웹 개발 최소 기능 제공, RESTful 요청 처리, 유니코드 기반, 필요한 부분은 추가해서 확장 가능
 - 참고: <http://flask.pocoo.org/>
- Flask 기본 사용법

1.1. Flask 모듈 설치 및 import

- `pip install flask`

```
1 from flask import Flask
```

1.2. Flask 객체를 app에 할당

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

```
1 __name__
```

```
'__main__'
```

▪ Flask 기본 사용법

▪ `__name__`

- `__name__` 이라는 변수는 모듈의 이름이 저장됨
- 실행하는 코드에서는 `__main__` 값이 들어감

▪ 파이썬과 시작점(entry point)

- 파이썬은 스크립트 언어
- 스크립트 언어는 전통적으로 시작점없이 스크립트 코드를 바로 실행함

▪ Flask 객체 생성

- `Flask(__name__)` 으로 설정하여, 현재 위치를 flask 객체에 알려줘야 함
 - 이름을 변경해도 정상 실행되지만, 일부 확장 기능 사용시에는 해당 이름을 정확히 알려주지 않을 경우 정상 동작되지 않음

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

```
1 app
```

<Flask '.__main__'>

1.3. 라우팅 경로를 설정

▪ 라우팅(route)

- 적절한 목적지를 찾아주는 기능, URL을 해당 URL에 맞는 기능과 연결해 줌
(예) `http://www.naver.com/hello`
 - `http://www.naver.com` 서버에서 `hello` 이라는 목적지에 맞는 함수를 호출해 줌

▪ URL

- Uniform Resource Locator
- 인터넷 상의 자원 위치 표기를 위한 규약
- WWW 주요 요소 중 하나: HTML, URL, HTTP

▪ URL vs URI

- URI(Uniform Resource Identifier): 통합 자원 식별자
- URL의 하위 개념이 URI
 - `https://www.naver.com` 주소
 - `https://www.naver.com` 이라는 서버를 나타내는 URL이면서 URI
 - `https://www.naver.com/input?id=grace&pw=1111` 주소
 - `https://www.naver.com/input` 은 URL
 - `https://www.naver.com/input?id=grace&pw=1111` 은 URI
 - 원하는 정보를 얻기 위해서는 `**?id=grace&pw=1111**` 라는 식별자가 필요하기 때문

1.3. 라우팅 경로를 설정

▪ 라우팅(route)

- 적절한 목적지를 찾아주는 기능, URL을 해당 URL에 맞는 기능과 연결해 줌
(예) `http://www.naver.com/hello`
 - `http://www.naver.com` 서버에서 `hello` 이라는 목적지에 맞는 함수를 호출해 줌

```
1 @app.route("/hello")
2 def hello():
3     return "<h1>Hello World!</h1>"
```

- @ 으로 시작하는 코드는 데코레이터라고 함

1.4. 메인 모듈로 실행될 때 flask 웹 서버 구동

- 서버로 구동한 IP 와 포트를 옵션으로 넣어줄 수 있음
- app.run() 함수로 서버 구동 가능
- host, port, debug 를 주로 사용함
 - host: 웹주소
 - port: 포트
 - debug: True or False
- app.run(host=None, port=None, debug=True)

■ 자신의 PC 에서 웹서비스 구현

- localhost, 127.0.0.1, 또는 0.0.0.0 으로 host 설정
- app.run() 함수로 자체 웹서버 구현 가능

```
1 host_addr = "127.0.0.1"
2 port_num = "5000"
```

```
1 if __name__ == "__main__":
2     app.run(host=host_addr, port=port_num)
```

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
```

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

1.5. 전체 기본 코드

```
1 from flask import Flask
2
3 app = Flask(__name__)
4 @app.route("/hello")
5 def test():
6     return "Hello Flask!"
7
8 if __name__ == "__main__":
9     app.run(host="127.0.0.1", port="5000")
```

← → ↻ ⓘ 127.0.0.1:5000

Hello, Flask~!

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

■ 코드 실행 방법

- flask 라이브러리를 사용한 코드는 보통 파일이름.py 로 작성한 후 => python 파일이름.py 으로 실행
- 서버에서 실행하는 것이 일반적
- 노트북에서는 실행 후, 테스트 끝나면, Terminate(■)를 눌러줘야 함

Flask와 Rest API

1. 정적 페이지 리턴하기

- 복잡한 URI를 함수로 쉽게 연결하는 방법 제공
- HTML 이용 : h1 ~ h6 는 HTML 헤드라인 태그

```
1 from flask import Flask
2
3 app = Flask(__name__)
4 @app.route("/")
5 def hello():
6     return "<h1>Hello World!</h1>"
7
8 @app.route("/first")
9 def hello_first():
10     return "<h1>Hello First!</h1>"
11
12 @app.route("/second")
13 def hello_second():
14     return "<h1>Hello Second</h1>"
15
16 if __name__ == "__main__":
17     app.run(host="127.0.0.1", port="5000")
```

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
```

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

← → ↻ ⓘ 127.0.0.1:5000

Hello World!

← → ↻ ⓘ 127.0.0.1:5000/first

Hello First!

← → ↻ ⓘ 127.0.0.1:5000/second

Hello Second

Flask와 Rest API

2. 복잡한 라우팅: 데이터 전달하기

- **URI를 변수로 사용**
 - 1) `http://127.0.0.1:8000/` 접속
 - 2) `http://127.0.0.1:8000/profile/grace` 접속
 - 3) `http://127.0.0.1:8000/first/grace` 접속

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def hello():
7     return "<h1>Hello World!</h1>"
8
9 @app.route("/profile/<username>")
10 def get_profile(username):
11     return "<h1>Profile: " + username + "!</h1>"
12
13 @app.route("/first/<username>")
14 def get_first(username):
15     return "<h1>First: " + username + "!</h1>"
16
17 if __name__ == "__main__":
18     app.run(host="127.0.0.1", port="5000")
```

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
```

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

← → ↻ ⓘ 127.0.0.1:5000

Hello World!

← → ↻ ⓘ 127.0.0.1:5000/profile/grace

Profile: grace!

← → ↻ ⓘ 127.0.0.1:5000/first/grace

First: grace!

Flask와 Rest API

3. 복잡한 라우팅: 데이터 전달하기 2

- **URI를 변수로 사용**, 변수에 데이터 타입도 줄 수 있음
 - 데이터 타입이 없으면 문자열로 인식
 - **int 이외에 float 도 데이터 타입으로 줄 수 있음**
 - `http://127.0.0.1:8000/message/1` 접속

← → ↻ ⓘ 127.0.0.1:5000

Hello World!

← → ↻ ⓘ 127.0.0.1:5000/message/100

message id: 100

← → ↻ ⓘ 127.0.0.1:5000/first/100

600

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 def add_file(data):
6     return data + 500
7
8 @app.route("/")
9 def hello():
10     return "<h1>Hello World!</h1>"
11
12 @app.route("/message/<int:message_id>")
13 def get_message(message_id):
14     return "message id: %d" % message_id  # %d 는 int, %f 는 float, %s 는 string
15
16 @app.route("/first/<int:messageid>")
17 def get_first(messageid):
18     data = add_file(messageid)
19     return "<h1>%d</h1>" % (data)
20
21 if __name__ == "__main__":
22     app.run(host="127.0.0.1", port="5000")
```

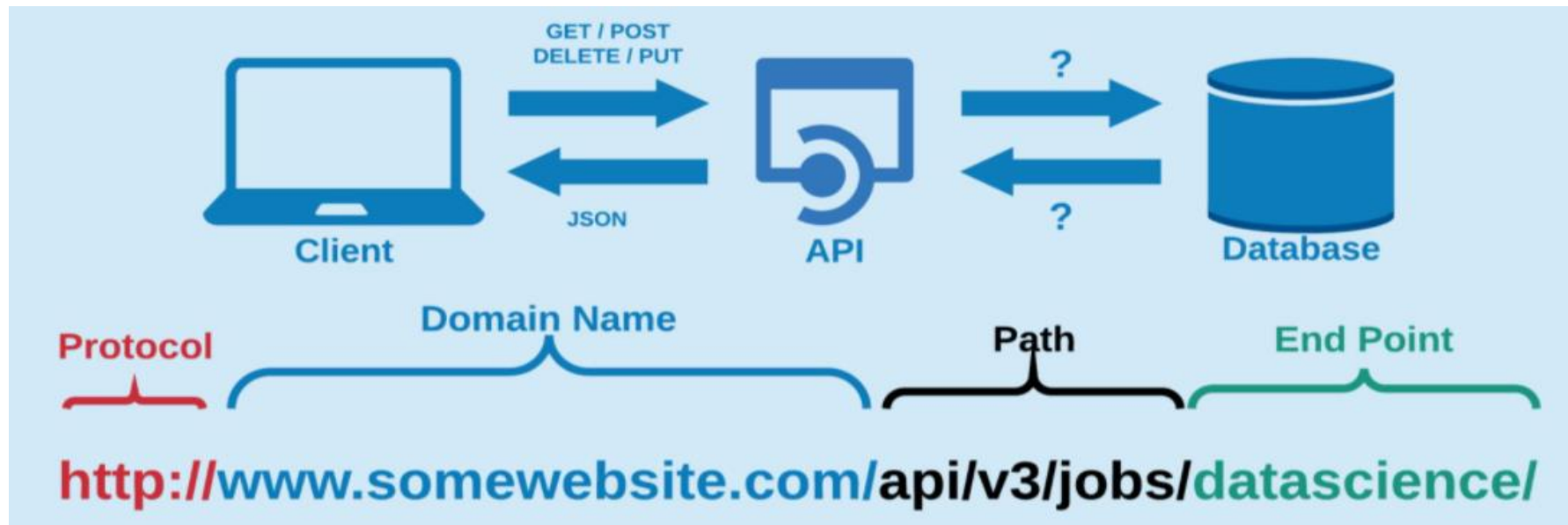
```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
```

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Flask와 Rest API

4. Flask로 REST API 구현하기

- HTTP(Hypertext Transfer Protocol)
 - Server/Client 모델로 Request/Response 사용
 - Client에서 요청(Request)을 보내면, Server에서 응답(Response)을 준다.
- 프로토콜 (protocol): 컴퓨터간 통신을 하기 위한 규칙



<https://towardsdatascience.com/launch-your-own-rest-api-using-flask-python-in-7-minutes-c4373eb34239>

Flask와 Rest API

4. Flask로 REST API 구현하기

- HTTP(Hypertext Transfer Protocol) Request/Response



<https://towardsdatascience.com/launch-your-own-rest-api-using-flask-python-in-7-minutes-c4373eb34239>

Flask와 Rest API

4. Flask로 REST API 구현하기

- HTTP(Hypertext Transfer Protocol) Request/Response
- Request

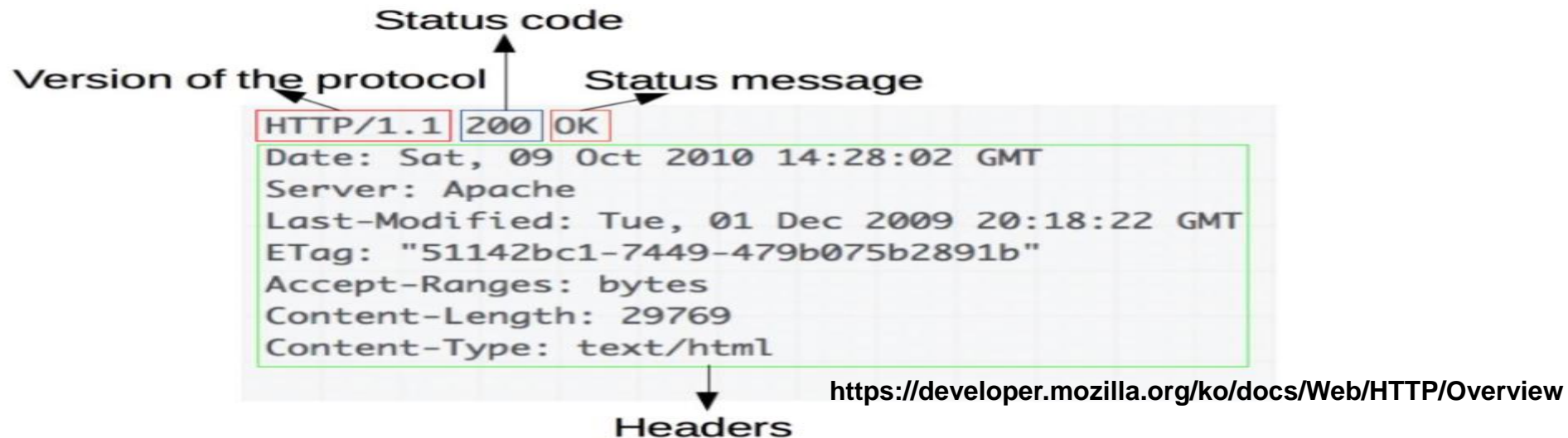


- **HTTP 메서드**
 - 클라이언트가 수행하고자 하는 동작을 정의한 GET, POST 같은 동사나 OPTIONS나 HEAD와 같은 명사
 - GET : 클라이언트는 리소스를 가져오는 동작
 - POST : HTML 폼의 데이터를 전송 하려고 사용하는 동작
- **Path - 가져오려는 리소스의 경로**
 - 예) 프로토콜 (http://), 도메인 (developer.mozilla.org), TCP 포트 (80)인 요소들을 제거한 리소스의 URL
- **Version of the protocol : HTTP 프로토콜의 버전**
- **Headers**
 - 서버에 대한 추가 정보를 전달하는 선택적 헤더들
 - POST와 같은 몇 가지 메서드를 위한 전송된 리소스를 포함하는 응답의 본문과 유사한 본문

Flask와 Rest API

4. Flask로 REST API 구현하기

- HTTP(Hypertext Transfer Protocol) Request/Response
- Response



- Version of the protocol : HTTP 프로토콜의 버전
- Status Code : 요청의 성공 여부와, 그 이유를 나타내는 상태 코드
- Status message : 아무런 영향력이 없는, 상태 코드의 짧은 설명을 나타내는 상태 메시지
- Headers
 - 요청 헤더와 비슷한, HTTP 헤더들
 - 선택 사항으로, 가져온 리소스가 포함되는 본문

Flask와 Rest API

4. Flask로 REST API 구현하기

▪ REST(Representational State Transfer)

- 자원(resource)의 표현(representation)에 의한 상태 전달
- HTTP URI를 통해 자원을 명시하고, HTTP Method를 통해 자원에 대한 CRUD Operation 적용
- **CRUD Operation와 HTTP Method**
 - Create: 생성(POST)
 - Read: 조회(GET)
 - Update: 수정(PUT)
 - Delete: 삭제(DELETE)

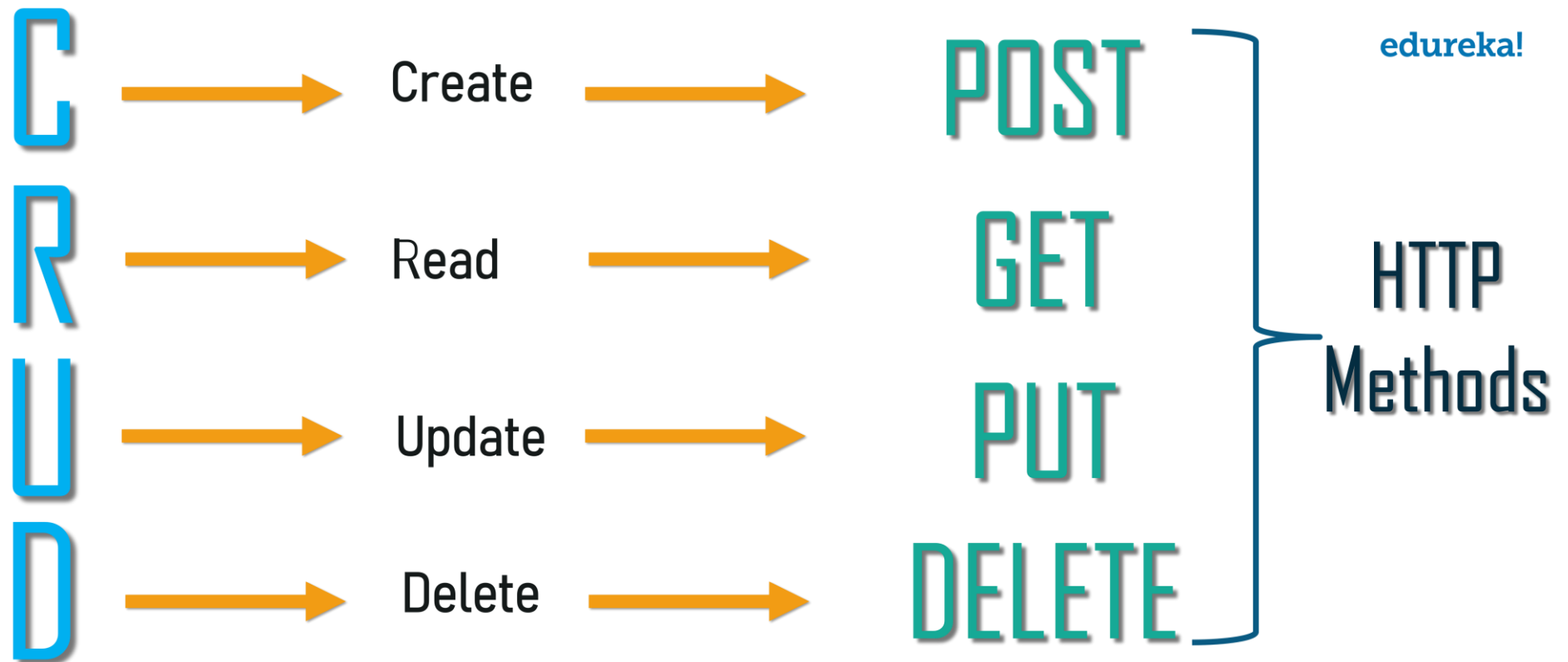
▪ REST API

- REST 기반으로 서비스 API를 구현한 것
- 마이크로 서비스, OpenAPI(누구나 사용하도록 공개된 API) 등에서 많이 사용됨

Flask와 Rest API

4. Flask로 REST API 구현하기

- HTTP Methods와 DB의 CRUD Operation

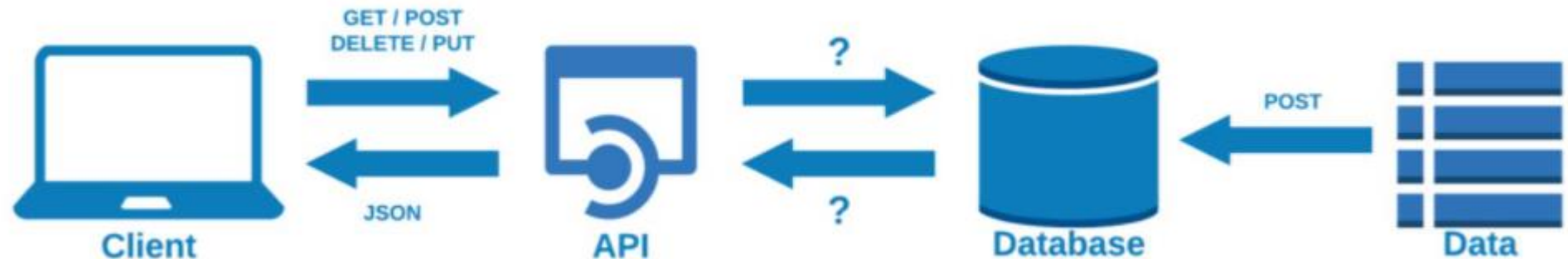


Flask와 Rest API

4. Flask로 REST API 구현하기

■ Flask 로 REST API 구현 방법

- 특정한 URI를 요청하면 JSON 형식으로 데이터를 반환하도록 만들면 됨
- 즉, 웹 주소(URI) 요청에 대한 응답(Response)를 JSON 형식으로 작성
- Flask에서는 dict(사전) 데이터를 응답 데이터로 만들고, 이를 jsonify() 메서드를 활용해서 JSON 응답 데이터로 만들 수 있음



<https://towardsdatascience.com/launch-your-own-rest-api-using-flask-python-in-7-minutes-c4373eb34239>

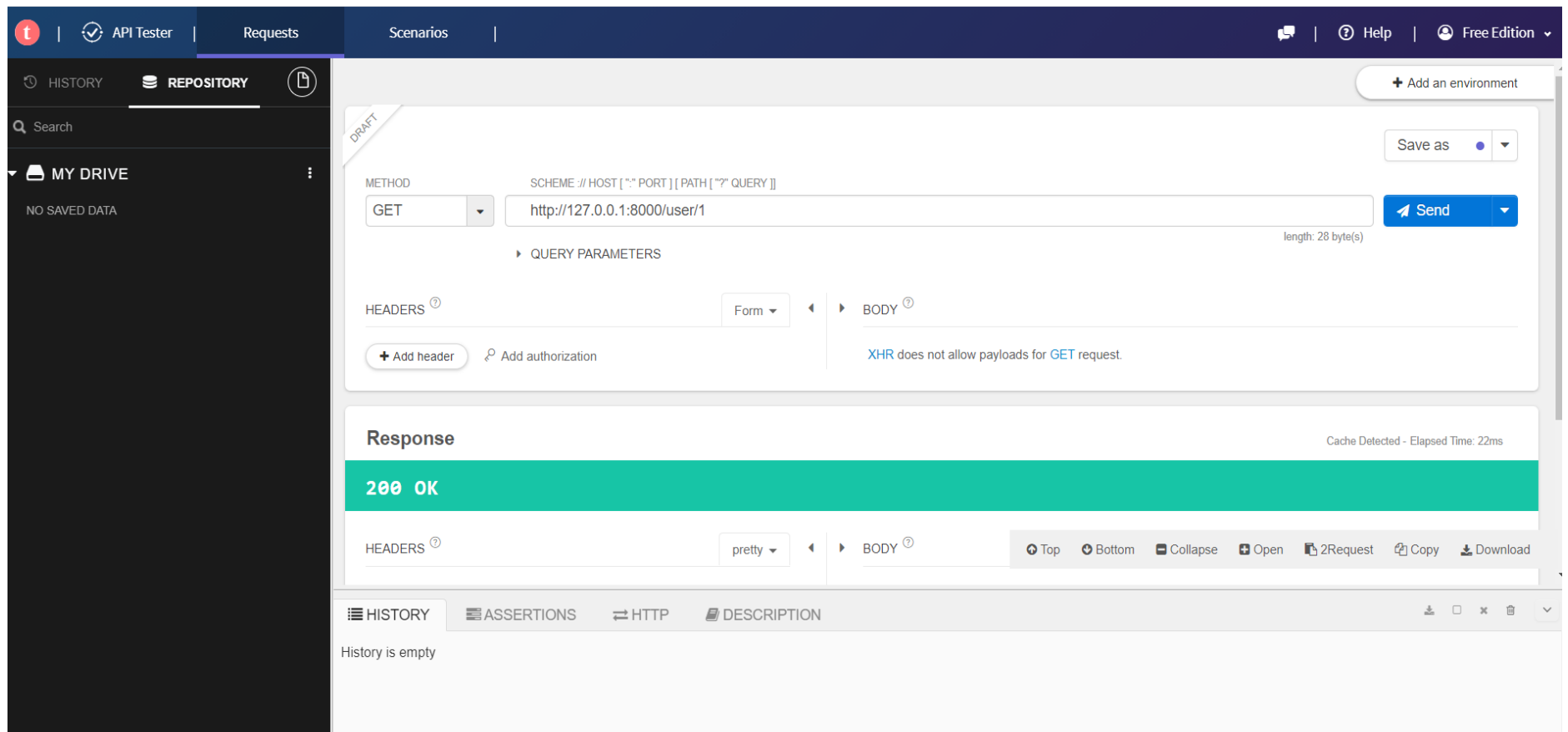
■ Flask의 jsonify() 함수

- 리턴 데이터를 JSON 포맷으로 제공

Flask와 Rest API

4. Flask로 REST API 구현하기

- REST API 테스트를 위한 준비
 - Talend API Tester 설치 - 크롬 브라우저 무료 확장 프로그램



Flask와 Rest API

4. Flask로 REST API 구현하기

▪ REST API 테스트

```
1 from flask import Flask, request, jsonify
2
3 app = Flask(__name__)
4
5 # 서버 리소스
6 resource = []
7
8 # 사용자 정보 조회
9 @app.route('/user/<int:user_id>', methods=['GET'])
10 def get_user(user_id):
11     for user in resource:
12         if user['user_id'] == user_id:
13             return jsonify(user)
14     return jsonify(None)
15
16 # 사용자 추가
17 @app.route('/user', methods=['POST'])
18 def add_user():
19     user = request.get_json()
20     resource.append(user)
21     return jsonify(resource)
22
23 if __name__ == "__main__":
24     app.run(host="127.0.0.1", port="8000")
```

Flask와 Rest API

4. Flask로 REST API 구현하기

REST API 테스트

The screenshot displays a REST client interface with the following components:

- Method:** A dropdown menu set to **POST**, highlighted with a red box and labeled "POST 선택".
- URL:** A text input field containing `http://127.0.0.1:8000/user`, highlighted with a red box and labeled "URL 입력".
- Headers:** A section with a checkbox for **Content-Type** set to `application/json`.
- Body:** A text area containing a JSON object:

```
{
  "user_id": 1,
  "name": "홍길동",
  "age": 28
}
```

, highlighted with a red box and labeled "json 형태로 입력".
- Response:** A green bar at the bottom indicating a **200 OK** status.

Additional interface details include a "Send" button, a "Save as" dropdown, and a "length: 26 byte(s)" indicator for the request body. The bottom status bar shows "Cache Detected - Elapsed Time: 14ms".

Flask와 Rest API

4. Flask로 REST API 구현하기

REST API 테스트

The screenshot shows a REST client interface with the following components:

- Method:** A dropdown menu set to **GET**, with a red arrow and the text "GET 선택" (GET selection) pointing to it.
- URL:** A text input field containing `http://127.0.0.1:8000/user/1`, with a red arrow and the text "URL 입력" (URL input) pointing to it.
- Buttons:** A "Save as" button and a "Send" button. The "Send" button is highlighted with a red box and a red arrow and the text "클릭" (click) pointing to it.
- Response:** A green bar at the bottom left displays **200 OK**, which is highlighted with a red box. The text "Response" is also highlighted with a red box.
- Body:** The response body is displayed in a red box with the following JSON data:

```
{  age : 28,  name : "홍길동",  user_id : 1}
```
- Headers:** The response headers are listed on the left:

```
Content-Type: application/json
Content-Length: 51 bytes
Server: Werkzeug/1.0.1 Python/3.7.10
Date: Mon, 15 Mar 2021 15:30:50 GMT
```
- Cache:** A status message at the top right says "Cache Detected - Elapsed Time: 18ms".
- Lengths:** The request length is "length: 28 byte(s)" and the response length is "length: 51 bytes".

Flask와 Rest API

5. REST API 구현

- data를 사전 데이터로 만들고, 이를 jsonify() 메서드에 넣어서 return

```
1 from flask import Flask, jsonify
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def hello():
7     return "<h1>Hello World!</h1>"
8
9 @app.route('/json_test')
10 def hello_json():
11     data = {'name' : 'Hong Gil Dong', 'city' : 'Seoul'}
12     return jsonify(data)
13
14 @app.route('/server_info')
15 def server_json():
16     data = { 'server_name' : '127.0.0.1', 'server_port' : '5000' }
17     return jsonify(data)
18
19 if __name__ == "__main__":
20     app.run(host="127.0.0.1", port="5000")
```

← → ↻ ⓘ 127.0.0.1:5000/json_test

{"city":"Seoul","name":"Hong Gil Dong"}

← → ↻ ⓘ 127.0.0.1:5000/server_info

{"server_name":"127.0.0.1","server_port":"5000"}

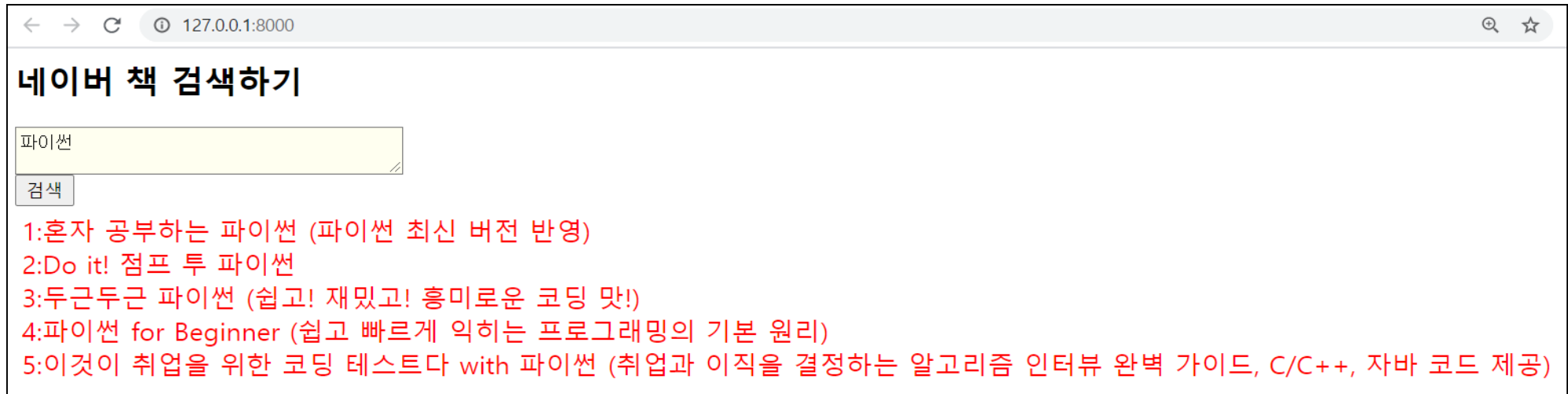
```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Flask와 Rest API

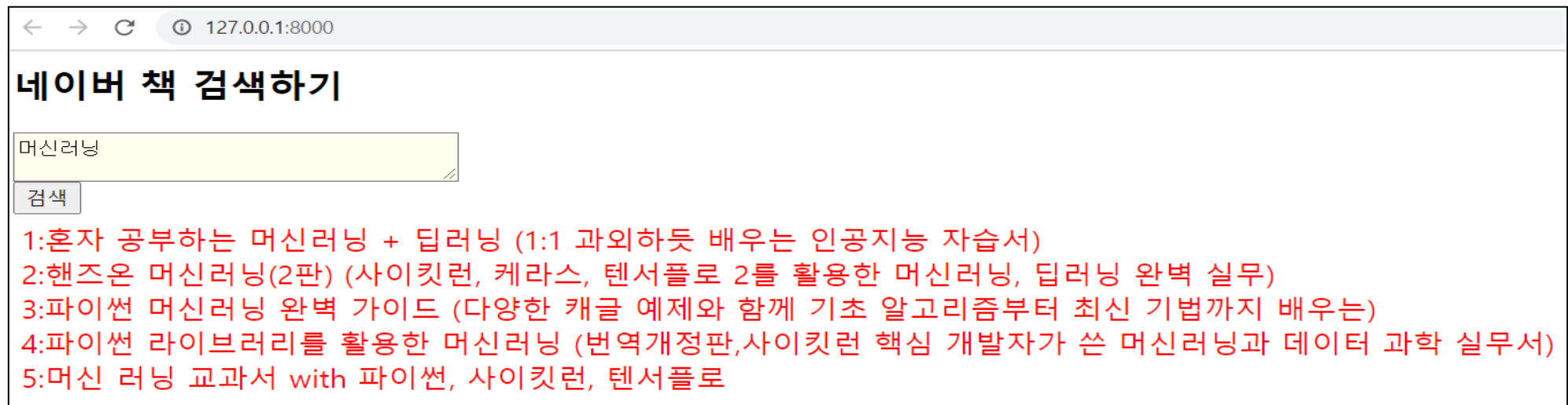
6. 네이버 OPEN API 책 검색과 Flask 연동

- 네이버 OPEN API 책 검색 + Flask 크롤링 결과 출력



A screenshot of a web browser window with the address bar showing '127.0.0.1:8000'. The page title is '네이버 책 검색하기'. Below the title is a search bar containing the text '파이썬' and a '검색' (Search) button. The search results are listed in red text:

- 1:혼자 공부하는 파이썬 (파이썬 최신 버전 반영)
- 2:Do it! 점프 투 파이썬
- 3:두근두근 파이썬 (쉽고! 재밌고! 흥미로운 코딩 맛!)
- 4:파이썬 for Beginner (쉽고 빠르게 익히는 프로그래밍의 기본 원리)
- 5:이것이 취업을 위한 코딩 테스트다 with 파이썬 (취업과 이직을 결정하는 알고리즘 인터뷰 완벽 가이드, C/C++, 자바 코드 제공)



A screenshot of a web browser window with the address bar showing '127.0.0.1:8000'. The page title is '네이버 책 검색하기'. Below the title is a search bar containing the text '머신러닝' and a '검색' (Search) button. The search results are listed in red text:

- 1:혼자 공부하는 머신러닝 + 딥러닝 (1:1 과외하듯 배우는 인공지능 자습서)
- 2:핸즈온 머신러닝(2판) (사이킷런, 케라스, 텐서플로 2를 활용한 머신러닝, 딥러닝 완벽 실무)
- 3:파이썬 머신러닝 완벽 가이드 (다양한 캐글 예제와 함께 기초 알고리즘부터 최신 기법까지 배우는)
- 4:파이썬 라이브러리를 활용한 머신러닝 (번역개정판,사이킷런 핵심 개발자가 쓴 머신러닝과 데이터 과학 실무서)
- 5:머신 러닝 교과서 with 파이썬, 사이킷런, 텐서플로

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

네이버 OPEN API를 활용한 책 검색

```
1 import requests
2
3 # 인증 정보
4 client_id = " "
5 client_secret = " "
6
7 # 기본 url 정보
8 url = "https://openapi.naver.com/v1/search/book.json"
9
10 # 검색할 책 이름 입력
11 q = "파이썬"
12 # url 호출 시 전달할 요청 변수 정보
13 params = {"query": q,          # 책이름
14           "display": 5,       # 검색할 건수
15           "sort": "count"}    # 정렬순서
16
17 # requests 라이브러리를 이용한 책 검색 api 호출
18 # get 방식으로 호출(url)/ 요청 변수 전달(params)/ 인증 정보 및 인코딩 정보 전달(header)
19 response = requests.get(url=url, params=params,
20                           headers={"X-Naver-Client-Id": client_id,
21                                   "X-Naver-Client-Secret": client_secret,
22                                   "Content-Type": "application/json; charset=utf-8"})
23
```


Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

```
24 # 호출 처리 상태 정보 rescode 변수에 할당
25 rescode = response.status_code
26
27 if (rescode == 200):
28     # 호출 처리 상태가 정상(200) 일 경우리턴 받은 책 정보 저장
29     data = response.json()
30 else:
31     print("Error Code:", rescode)
32
33 # Naver 책 검색 API 응답 중 실제 책 아이템 데이터 추출 및 출력
34 # print(data)
35 item_list = data["items"]
36
37 # 검색한 도서를 리스트 형태로 저장하여 리턴하기
38 book_list = []
39 for item in item_list:
40     book_list.append(item["title"].replace("<b>", "").replace("</b>", "")+"<br>")
41
42 print(book_list)
43
```

['혼자 공부하는 파이썬 (파이썬 최신 버전 반영)
', 'Do it! 점프 투 파이썬
', '두근두근 파이썬 (쉽고! 재밌고! 흥미로운 코딩 맛!)
', '파이썬 for Beginner (쉽고 빠르게 익히는 프로그래밍의 기본 원리)
', '이것이 취업을 위한 코딩 테스트다 with 파이썬 (취업과 이직을 결정하는 알고리즘 인터뷰 완벽 가이드, C/C++, 자바 코드 제공)
']

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

Flask와 네이버 책 검색 연동

- 1) flask 구동 파이썬 파일
- 2) index.html

```
1 from flask import Flask, request, jsonify
2 import pprint
3 import requests
4 import json
5
6 app = Flask(__name__)
7
8 # 루트에 접근할 경우
9 @app.route('/', methods=['GET'])
10 def index():
11     with open("index.html", "rb") as f:
12         return f.read()
13
```

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

```
14 @app.route('/api', methods=['GET', 'POST'])
15 def search_book():
16     request_json = request.json
17
18     # 검색할 책 이름을 index.html에서 받아오기
19     q = request.args.get('q', '')
20     if q == '':
21         return '{"책 이름": "조회할 책 이름을 입력해주세요!!"}'
22     print("q=", q)
23
24     # 네이버 OPEN API 인증 정보
25     client_id = " "
26     client_secret = " "
27
28     # 네이버 도서명 검색 url 정보
29     url = "https://openapi.naver.com/v1/search/book.json"
30
31     # url 호출 시 전달할 요청 변수 정보
32     params = {"query": q,          # 책이름
33              "display": 5,        # 조회 건수
34              "sort": "count"}    # 정렬 순서
35
```

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

```
36 # requests 라이브러리를 이용한 책 검색 api 호출
37 # get 방식으로 호출(url)/ 요청 변수 전달(params)/ 인증 정보 및 인코딩 정보 전달(header)
38 response = requests.get(url=url, params=params,
39                          headers={"X-Naver-Client-Id": client_id,
40                                  "X-Naver-Client-Secret": client_secret,
41                                  "Content-Type": "application/json; charset=utf-8"})
42 # 호출 처리 상태 정보 rescode 변수에 할당
43 rescode = response.status_code
44
45 if (rescode == 200):
46     # 호출 처리 상태가 정상(200) 일 경우리턴 받은 책 정보 저장
47     #print(response.json())
48     data = response.json()
49 else:
50     print("Error Code:", rescode)
51
52 # Naver 책 검색 API 응답 중 실제 책 아이템 데이터 추출 및 출력
53 # print(data)
54 item_list = data["items"]
55
```

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

```
56     # 검색한 도서를 리스트 형태로 저장하여 리턴하기
57     book_list = []
58     for item in item_list:
59         book_list.append(item["title"].replace("<b>", "").replace("</b>", "")+"<br>")
60     #print(book_list)
61
62     # 결과값 리턴하기
63     res = ""
64     for i, book in enumerate(book_list, start=1):
65         res += str(i) + ":" + book
66     return jsonify(res)
67
68 if __name__ == "__main__":
69     # 서버 실행하기
70     app.run(host="127.0.0.1", port="8000")
71
```

- * Serving Flask app "__main__" (lazy loading)
 - * Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 - * Debug mode: off
- * Running on <http://127.0.0.1:8000/> (Press CTRL+C to quit)

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

index.html

```
1 <DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5 </head>
6 <body>
7   <h2>네이버 책 검색하기</h2>
8   <div>
9     <textarea id="q" rows="2" cols="40"></textarea>
10    <br>
11    <button id="qButton">검색</button>
12    <div id="result"></div>
13  </div>
14  <script>
15    const qs = (q) => document.querySelector(q)
16    window.onload = () => {
17      const q = qs('#q')
18      const qButton = qs('#qButton')
19      const result = qs('#result')
```

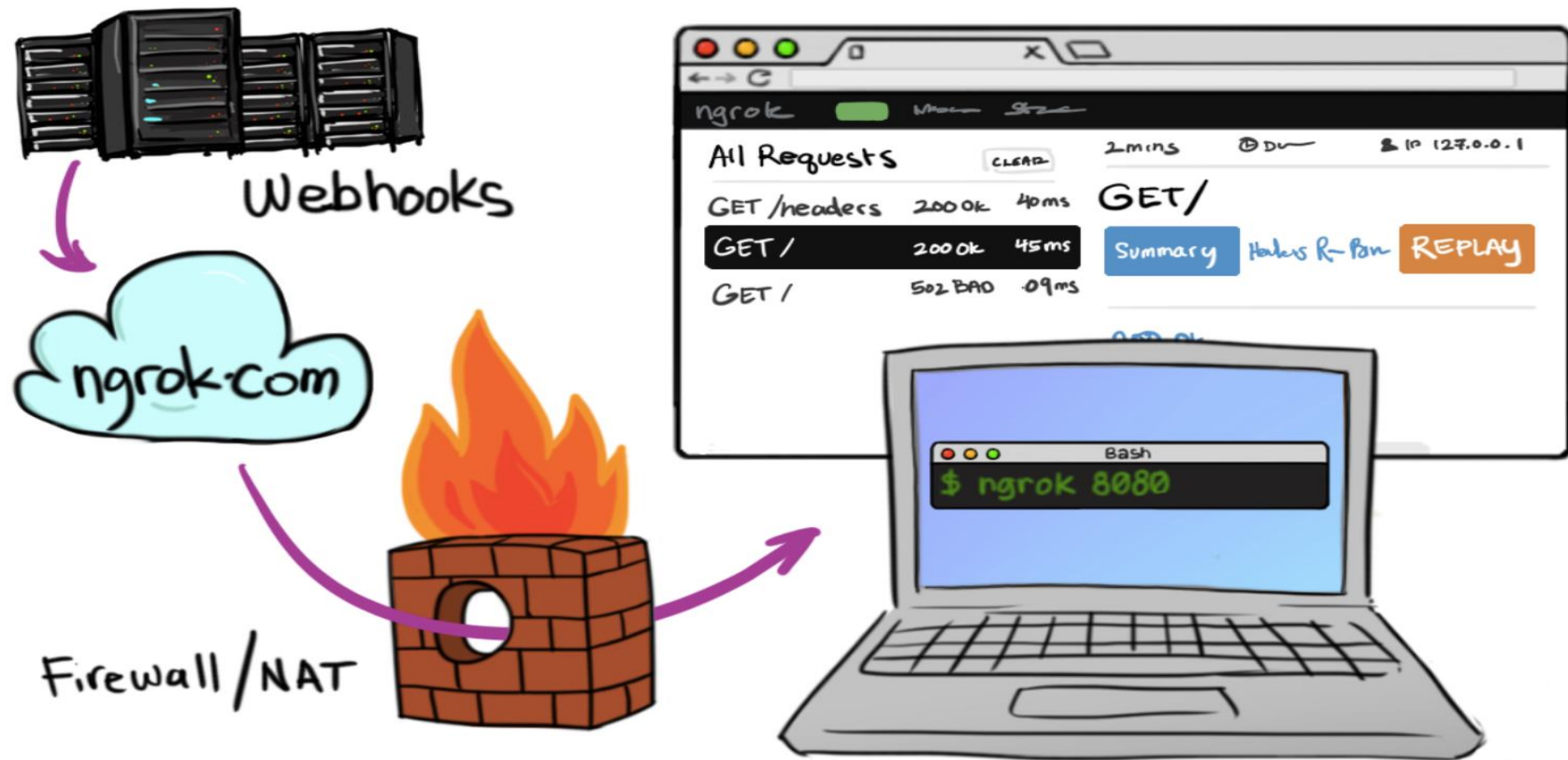
Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

```
20 // 조회 버튼을 눌렀을 때
21 qButton.onclick = () => {
22     result.innerHTML = "...";
23     // URL 생성하기
24     const api = "/api?q=" + encodeURIComponent(q.value)
25     // API에 접근하기
26     fetch(api).then((res) => {
27         return res.json() // JSON 응답
28     }).then((data) => {
29         // 결과를 화면에 출력하기
30         result.innerHTML =
31             data
32     })
33 }
34 }
35 </script>
36 <style>
37     #result {
38         padding: 5px;
39         font-size: 1.2em;
40         color: red;
41     }
42
43     #q {
44         background-color: #fffff0;
45     }
46 </style>
47 </body>
48 </html>
```

Flask와 Rest API

- Ngrok
 - 방화벽 넘어서 외부에서 로컬에 접속 가능하게 하는 터널 프로그램



Flask와 Rest API

▪ Ngrok 설치

- <https://ngrok.com/download>
- zip 파일을 받아 압축을 풀고 ngrok.exe라는 파일이 있는 위치에서 실행한다.
- > **ngrok.exe http 8000**

선택 Anaconda Prompt (anaconda3) - ngrok.exe http 8000

ngrok by @inconshreveable

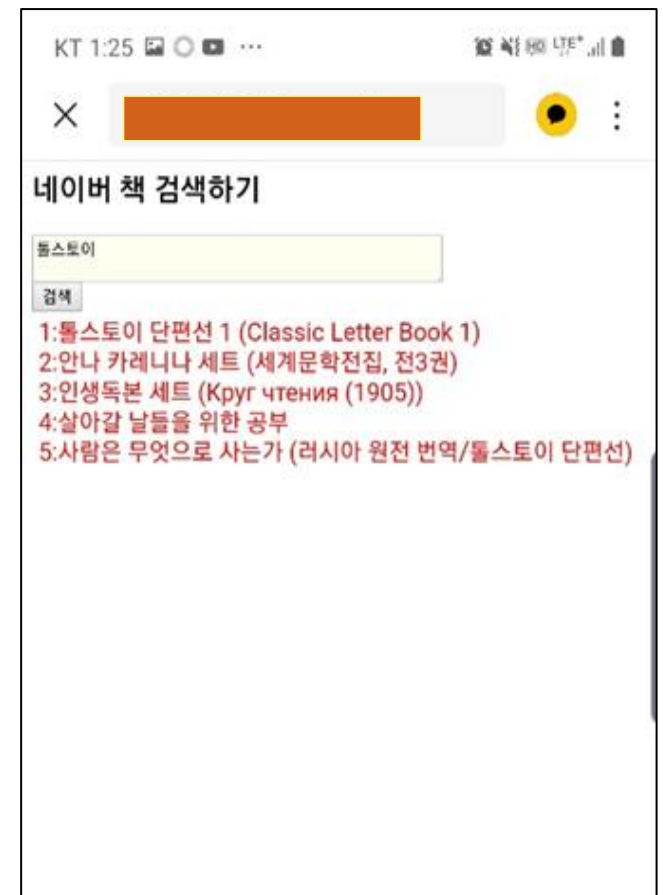
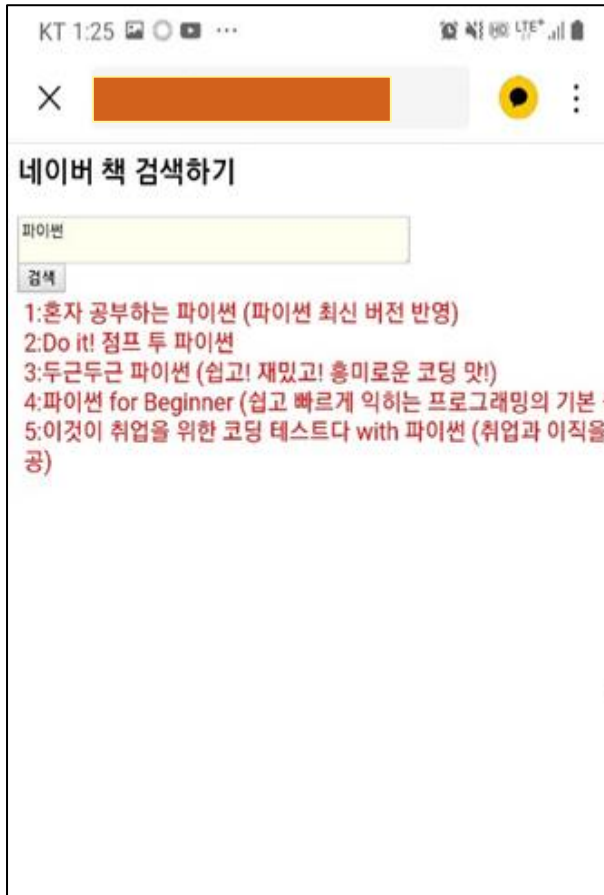
```
Session Status      online
Account             Kim Jin Suk (Plan: Free)
Version             2.3.35
Region             United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://[redacted] -> http://localhost:8000
Forwarding           https://[redacted] -> http://localhost:8000

Connections         ttl    opn    rt1    rt5    p50    p90
                   15     0      0.00   0.00   0.34   0.68
```

Flask와 Rest API

6. 네이버 OPEN API 책 검색과 Flask 연동

- 네이버 OPEN API 책 검색 + Flask 크롤링 결과 출력



Flask와 Rest API

7. 프론트엔드와 백엔드 Flask 로 한번에 구현해보기

- 폴더 구조

```
flask_test 폴더
  /login.py
  /templates 폴더
    /login.html
  static 폴더
```

- login.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form action="/login" method="get">
      <p>Enter Name:</p>
      <p><input type="text" name="user_name" /></p>
      <p><input type="submit" value="submit" /></p>
    </form>
  </body>
</html>
```

Flask와 Rest API

7. 프론트엔드와 백엔드 Flask 로 한번에 구현해보기

- login.py 파이썬 코드
 - GET 요청으로 받는 url은 아래와 같이 코드 작성

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

```
@app.route('/login')
def login():
    username = request.args.get('user_name')
    if username == 'grace':
        return_data = {'auth':'success'}
    else:
        return_data = {'auth':'failed'}
    return jsonify(return_data)
```

```
if __name__ == '__main__':
    app.run(host="127.0.0.1", port="8000")
```

Flask와 Rest API

7. 프론트엔드와 백엔드 Flask 로 한번에 구현해보기

- Rest API 요청시 파라미터/파라미터값 넣기
 - HTTP 의 요청 방식 중, 가장 많이 사용되는 방식이 **GET** 방식
 - GET 방식에서는 URI 상에 파라미터와 파라미터 값을 넣을 수 있음
 - 규칙: `URL?파라미터1=파라미터1값&파라미터2=파라미터2값`
 - URL 이후 첫 파라미터 이름 전에 ? 를 표시하고, 추가 파라미터가 있을 시에는 & 표시를 해야 함
 - (예) http://localhost:8080/login?user_name=grace&pwd=1111
- Flask 로 정적 웹페이지 로드하기
 - 프론트엔드 페이지도 flask 로 보여줄 수 있음
 - `flask render_template(HTML파일명)`: HTML 파일 전송하기
 - HTML파일은 flask 가 실행되는 하위 폴더인 `templates` 폴더 안에 위치해야 함

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

```
def hello_html():
```

```
    # html file은 templates 폴더에 위치해야 함
```

```
    return render_template('login.html')
```

Flask와 Rest API

7. 프론트엔드와 백엔드 Flask 로 한번에 구현해보기

- login.py 파일 업데이트
 - flask 의 render_template 함수 추가
 - @app.route('html_test')@app.route('/html_test')

```
def hello_html():  
    # html file은 templates 폴더에 위치해야 함  
    return render_template('login.html')
```

Flask와 Rest API

7. 프론트엔드와 백엔드 Flask 로 한번에 구현해보기

- login.py(완성)

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

```
@app.route('/login')
def login():
    username = request.args.get('user_name')
    if username == 'grace':
        return_data = {'auth': 'success'}
    else:
        return_data = {'auth': 'failed'}
    return jsonify(return_data)
@app.route('/html_test')
def hello_html():
    # html file은 templates 폴더에 위치해야 함
    return render_template('login.html')
```

```
if __name__ == '__main__':
    app.run(host="127.0.0.1", port="8000")
```

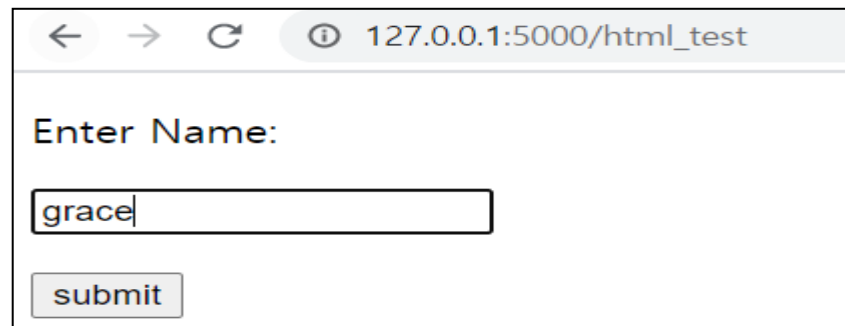
Flask와 Rest API

7. 프론트엔드와 백엔드 Flask 로 한번에 구현해보기

■ 코드 실행하기

- 1) >python login.py ----- login.py 실행
- 2) http://127.0.0.1:8000/html_test ----- 웹페이지 브라우저로 실행

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



← → ↻ ⓘ 127.0.0.1:5000/html_test

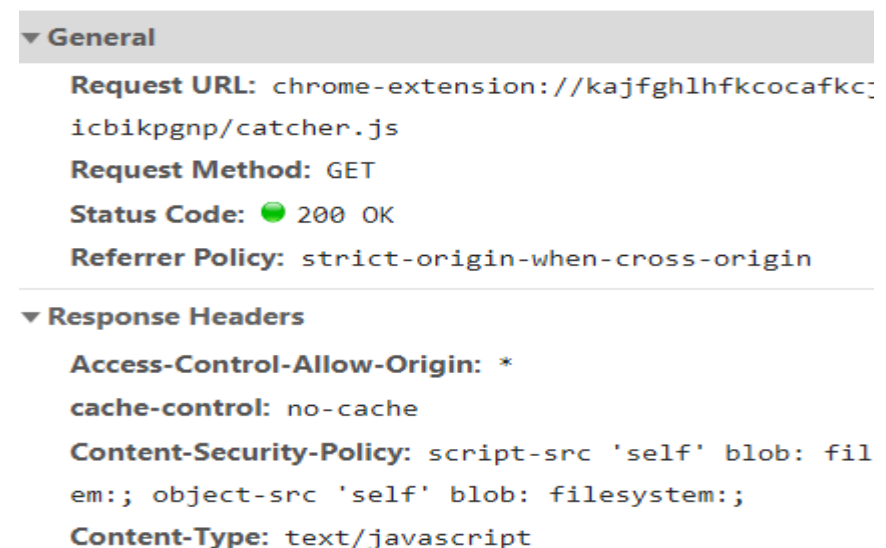
Enter Name:

submit



← → ↻ ⓘ 127.0.0.1:5000/login?user_name=grace

```
{"auth": "success"}
```



▼ General

Request URL: chrome-extension://kajfghlhfkcoafkc:icbikpgnp/catcher.js

Request Method: GET

Status Code: 200 OK

Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers

Access-Control-Allow-Origin: *

cache-control: no-cache

Content-Security-Policy: script-src 'self' blob: filem;; object-src 'self' blob: filesystem;;

Content-Type: text/javascript