

# 빅데이터를 지탱하는 기술

## Week 1

### Clean code & API

23.07.19 / 엄소은

# CONTENTS

## 01. Clean Code

- python 팁들

## 02. 프로젝트 정리하기

- 깔끔한 프로젝트 구조
- Readme 작성법

## 03. API

- API 개념
- REST API
- 직접 API 써보기

## Clean code 작성하기

- 개발자도 아닌데 Clean code 작성법까지 알아야 하나 ?
- Data science 업무 = 코드 작성도 포함되어 있다 !!!
- DS 가 코드 작성 능력도 있으면 금상첨화 !!! 학회 프로젝트 할 때도 코드 작성 능력은 필수이다
- 이왕 할 거 clean code 방식으로 작성하자

(개발을 위한 것이 아니라 자세한 clean code 규칙(solid) 는 생략 -> 관심 있으면 Clean Architecture, 로버트 C.마틴 추천 !)

## 파이썬 팁1 – 가상환경

- `conda create -n <가상환경 이름> python==3.8`
- 제발 프로젝트 마다 가상환경 다르게 하자.
- `requirements.txt` 를 뽑을 때 내가 사용한 dependency 만 확인할 수 있다. (안 그러면 내가 사용 안 한 것도 다 짬뽕되는데, 받는 사람 입장에서 되게 귀찮고 번거롭다 )
- `pip freeze` 대신 `pipreqs` 사용 !!! (`pip install pipreqs`) -> 내가 사용한 library 만 `requirements.txt` 로 만들어준다  
`pipreqs path/to/project`

## 파이썬 팁2 – ArgumentParser

- main.py 를 실행할때, parameter 를 추가해서 실행하고 싶다면 ?
- ex) python main.py --text input.txt --ver kor 등등
- ArgumentParser 를 이용한다 !

```
"""
1. ArgumentParser 설정
"""
# ArgumentParser 객체 생성
parser = argparse.ArgumentParser(description='이 프로그램의 설명')

# 인자 추가하기
parser.add_argument('--text', type=str, required=True, default=None, help='input text 값')
parser.add_argument('--env', required=False, default='dev', help='실행환경은 뭐냐')

conf = parser.parse_args()

if conf.text is None:
    print("\n No Text is supplied. Please input text. \n")
elif conf.text.endswith(".txt"):
    with open(conf.text, 'r', encoding='utf-8') as f:
        text = f.read()
```

## 파이썬 팁 3 – logging

- 중요도에 따라 결과물을 출력하면서 보고 싶다 -> logging 사용

상황	방법
일반적인 console 출력	<code>print()</code>
프로그램의 실행 중 발생하는 정상적인 이벤트 알림	<code>logging.info()</code> (진단 등을 위한) 자세한 수준의 로그인 경우에는 <code>logging.debug()</code>
런타임 중 발생한 이벤트와 관련하여 경고	사용자가 프로그램을 수정해서 문제를 해결할 수 있는 경우 <code>warnings.warn()</code> 사용자가 처리할 수 있는 문제가 아닌 경우 <code>logging.warning()</code>
런타임 중 발생한 이벤트와 관련한 에러	예외 처리 ( <code>raise Exception</code> )
발생한 예외를 suppress하고 raise 하지 않은 경우 (e.g. long-running 서버 프로세스에서 에러 발생 시)	<code>logging.error()</code> , <code>logging.exception()</code> , <code>logging.critical()</code>

## 파이썬 팁 3 – logging

- 출력하고 싶은 레벨을 세팅할 수 있다 ! (logging 단계: DEBUG < INFO < WARNING < ERROR < CRITICAL)
- 기본 레벨 = warning
- `logger.setLevel(logging.INFO)` 로 하면 어떤 것이 출력될까 ?

Level	Value	When to use
DEBUG	10	(주로 문제 해결을 할 때 필요한) 자세한 정보.
INFO	20	작업이 정상적으로 작동하고 있다는 확인 메시지.
WARNING	30	예상하지 못한 일이 발생하거나, 발생 가능한 문제점을 명시. (e.g. 'disk space low') 작업은 정상적으로 진행.
ERROR	40	프로그램이 함수를 실행하지 못 할 정도의 심각한 문제.
CRITICAL	50	프로그램이 동작할 수 없을 정도의 심각한 문제.

-> 중요도에 따라 출력을 다르게 할 수 있다 !

warning 이 날 것 같은 위치 :

`logger.Warn(경고)`

결과물이 궁금 :

`logger.info(결과물)`

등등 내가 다르게 설정할 수 있다 !

## 파이썬 팁 3 – logging

- Loggers : 어플리케이션 코드가 직접 사용할 수 있는 인터페이스를 제공합니다.
- Handlers : Loggers에 의해 생성된 LogRecord를 처리하여 적절한 위치로 보냅니다.
- Filters : 출력되어야 하는 로그를 필터링합니다.
- Formatters : LogRecord의 출력 형태를 지정합니다.

```
"""
2. Logger 설정
"""
# logger 객체 생성
logger = logging.getLogger(name='MyLog')
logger.setLevel(logging.INFO) ## 경고 수준 설정

# log 의 format
formatter = logging.Formatter('%(asctime)s|%(name)s|%(levelname)s|%(message)s',
                               datefmt='%Y-%m-%d %H:%M:%S')

stream_handler = logging.StreamHandler() ## 스트림 핸들러 생성
stream_handler.setFormatter(formatter) ## 텍스트 포맷 설정
logger.addHandler(stream_handler) ## 핸들러 등록

# log file 에 로그들을 모두 저장하고 싶을 때
logging.basicConfig(filename='myinfo.log', level=logging.INFO)
```



## 파이썬 팁 4 – config.yaml

- 바꿔서 테스트 해보고 싶은 변수들은 config.yaml 파일에 따로 저장해두자 !!
- (test-train split ratio, activation function, hidden layer 수, drop out rate 등등 )
- yaml 파일 관리를 위해 pyyaml 설치 (pip install pyyaml)

```
#INITIAL SETTINGS
data_directory: ./data/
data_name: breast-cancer-wisconsin.data
drop_columns: ["id"]
target_name: class
test_size: 0.2
model_directory: ./models/
model_name: KNN_classifier.pkl

#kNN parameters
n_neighbors: 5
weights: uniform
algorithm: auto
leaf_size: 15
p: 2
metric: minkowski
n_jobs: 1
```

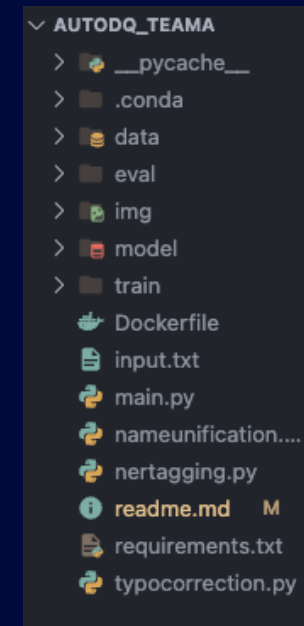
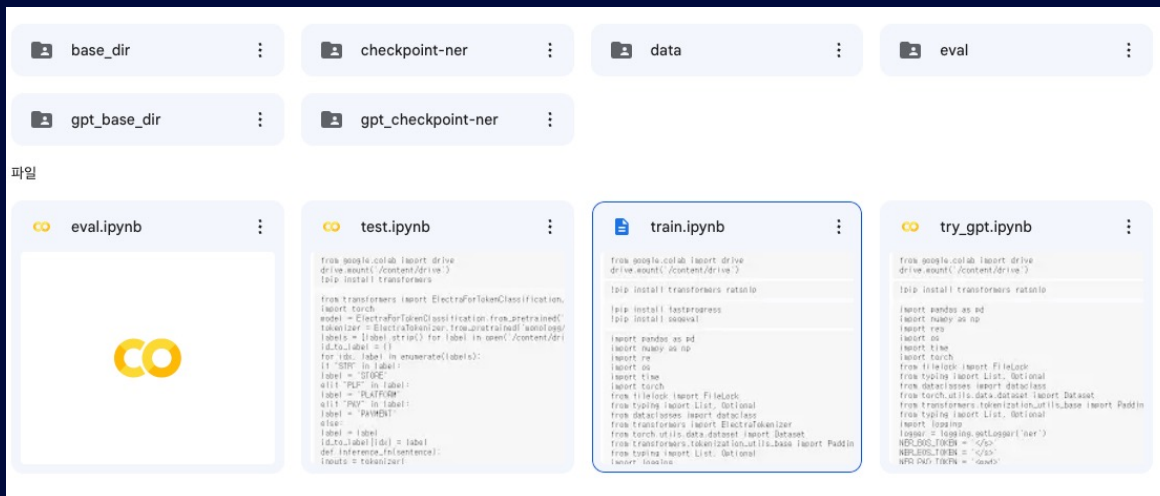
## 파이썬 실습

1. 가상환경 만들기 (python 3.8)
2. requirements.txt 설치
3. huggingface pipeline 이용해서 log 찍어가며 여러 파일 결과 확인하기
4. config.yaml 파일 내 변수 변경시키면서 train.py 파일 실행해보기

## 2. 프로젝트 정리하기

### 코랩에서 정리 끝 ≠ 프로젝트 끝

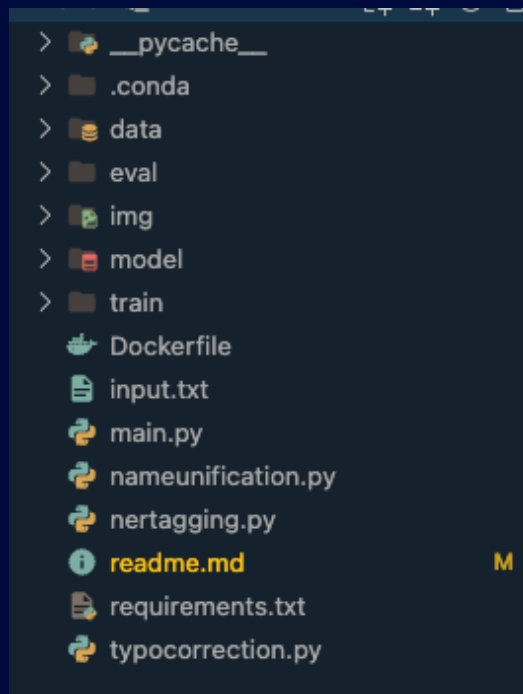
- Colab 에서 작업한 것이 끝이 아니다!  
-> 하나의 root 폴더 안에 깔끔하게 정리하고 Readme 작성까지 해야 프로젝트가 완전히 끝난 것
- 잘 정리된 하나의 프로젝트 >>>> 설명 없이 단순히 실행한 ipynb 파일 여러 개 있는 프로젝트
- 학교에서야 프로젝트로 끝나지만, 현업에서는 계속 유지보수 하려면 내가 쓴 코드가 남들도 읽기 쉬워야 한다!



## 2. 프로젝트 정리하기

### 프로젝트 파일 구조

- 하나의 file 내에 모든 것이 담겨 있어야 한다.
- 하나의 기능을 담당하는 것마다 .py 파일로 개별적으로 정리한다.
- main.py 파일은 다른 .py 파일들을 불러와서 모든 workflow를 합치는 역할을 해야 한다.



## 2. 프로젝트 정리하기

### 프로젝트 파일 구조

```
> .__pycache__
> .conda
└─ data
   ├── All_stores.csv
   ├── All_stores.txt
   ├── label_map.txt
   ├── new_pay.txt
   ├── new_platform.txt
   ├── new_stores.txt
   ├── pay_names.txt
   ├── platform_names.txt
   ├── train.csv
   └── val.csv
└─ eval
   ├── eval.py
   └── eval.txt
> img
└─ model
   ├── config.json
   ├── pytorch_model.bin
   ├── tokenizer_config.json
   └── vocab.txt
└─ train
   ├── data_labeling.py
   ├── train.py
   ├── utils.py
   ├── Dockerfile
   ├── input.txt
   ├── main.py
   ├── nameunification.py
   ├── nertagging.py
   ├── readme.md
   ├── requirements.txt
   └── typocorrection.py
```

- data : 모든 데이터 저장
- eval: eval 하는데 필요한 코드
- train: 모델 train 하는데 필요한 코드
- model: 모델과 관련된 모든 것들
- 그 외 모듈들은 root directory 에 저장

## 2. 프로젝트 정리하기

### 리드미 작성법

- 내 프로젝트를 자세히 설명해야 한다 !!
- 남들이 내 github를 보고 실행까지 할 수 있어야 한다.
- 가장 중요한 부분 -> end-to-end inference (실행설명서)

## 2. 프로젝트 정리하기

### Readme 작성법

#### Dependencies

- Python 3.8
- Pytorch 2.0.1
- dependencies in requirements.txt

#### How to run

##### 1) Using python file

1. Install pytorch and other dependencies

```
pip install -r requirements.txt
```

2. Run with options -For example,

```
python3 main.py --text input.txt
```

`input.txt` : Raw 결제내역 데이터

#### Result

- result\_dict
  - sentence: 원본 데이터
  - STORE: 인식한 스토어명
  - PLATFORM: 인식한 플랫폼명
  - PAY: 인식한 결제수단명
  - COST: 인식한 코스트명
- corrected\_dict: 오타수정, 같은 의미 단어 통일한 것

##### 2) Using Docker

```
docker pull soeunuhm/autodq:latest
```

```
docker run -v <Your-repository-name>:/app/input -it python-nf python main.py --text /app/input/input.txt
```

## 2. 프로젝트 정리하기

### Readme 작성법

#### File Description

##### main

- `main.py` : 전체 end-to-end pipeline 수행
- `ner tagging.py` : NER model 사용
- `nameunification.py` : 이름 통일
- `typocorrection.py` : 오타 정제

##### data

- `train.csv` : train data
- `val.csv` : val data

##### eval

- `eval.py` : NER model 평가

##### model

- `config.json` : model setting
- `pytorch_model.bin` : NER model
- `tokenizer.json` : tokenizer
- `vocab.txt` : tokenizer vocab

##### train

- `train.py` : NER model train
- `data_labeling.py` : 데이터 라벨링 하는데 필요한 함수
- `utils.py` : train 에 필요한 utils

내가 올린 파일들에 대한 설명도 자세히 !!



## API : Application Programming Interface

### • 점원의 역할



### • API의 역할



### <음식점>

손님이 점원에게 주문

점원은 요리사에게 주문 전달

-> API 는 "점원"의 역할

### <웹>

프론트엔드 (유저에게 보여지는 화면) <-> 백엔드(유저 데이터 저장) 간의 소통이 필요

이때 서로 상호작용하게 도와주는 것이 API



## API 의 역할

### 1. API는 서버와 데이터베이스에 대한 **출입구 역할**을 한다.

데이터베이스에는 소중한 정보들이 저장된다. 모든 사람들이 이 데이터베이스에 접근할 수 있으면 안 된다!

API는 이를 방지하기 위해 여러분이 가진 서버와 데이터베이스에 대한 출입구 역할을 하며, 허용된 사람들에게만 접근성을 부여해준다.

### 2. API는 애플리케이션과 기기가 원활하게 **통신**할 수 있도록 한다

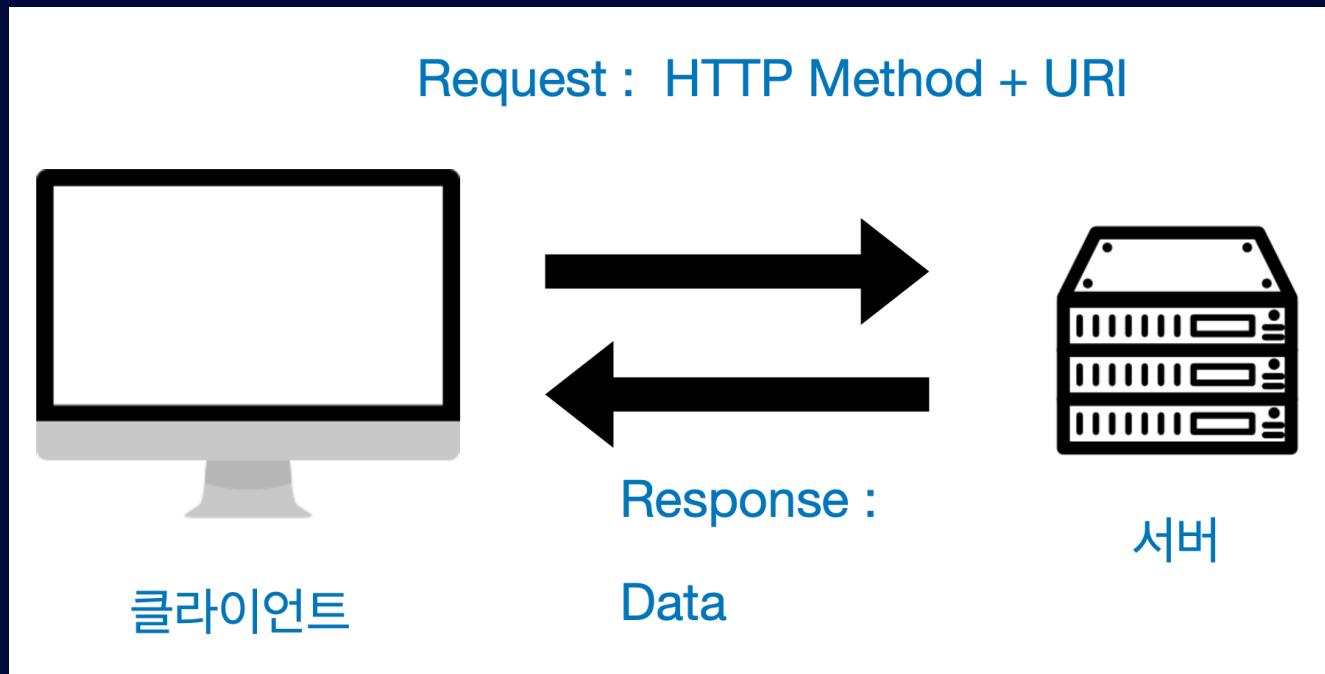
여기서 애플리케이션이란 우리가 흔히 알고 있는 스마트폰 어플이나 프로그램을 말한다. API는 애플리케이션과 기기가 데이터를 원활히 주고받을 수 있도록 돕는 역할을 한다.

### 3. API는 모든 접속을 **표준화**한다.

API는 모든 접속을 표준화하기 때문에 기계/ 운영체제 등과 상관없이 누구나 동일한 액세스를 얻을 수 있다. 쉽게 말해, API는 범용 플러그처럼 작동한다고 볼 수 있다.

# 3. API

그래서 어떻게 쓰는데?



## HTTP Method

Client

Server

GET  
POST  
PUT  
DELETE

## HTTP (Request) Method: Hypertext Transfer Protocol Method

클라이언트와 서버 간의 통신을 원활하게 하도록 세운 규칙

### HTTP Methods

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS
- CONNECT
- TRACE

- GET : 어떤 리소스로부터 특정 데이터를 읽거나(read) 검색(retrieve)할 때 사용
- POST: 서버에 있는 데이터베이스에 어떤 새로운 데이터를 추가(생성=create)하기 위해 사용
- PUT: 기존에 있는 데이터를 업데이트 하기 위해 사용
- DELETE: 기존에 있는 데이터를 삭제

# 3. API

## REST API

- REST = **R**epresentational **S**tate **T**ransfer
- REST란 어떤 자원에 대해

**CRUD**(Create, Read, Update, Delete) 연산을 수행하기 위해 **URI(Resource)**로 요청을 보내는 것으로, **Get, Post** 등의 방식(**HTTP Method**)을 사용하여 요청을 보낸다.

### CRUD

CRUD	요청의 뜻
Create	올려줘
Read	불러와줘
Update	바꿔줘
Delete	지워줘

데이터를 다룰 때 큰 틀의 기준이 되는 4가지 요청.  
대부분의 컴퓨터 소프트웨어가 가지는 기본적인 데이터 처리  
기능인 Create(생성), Read(읽기), Update(갱신),  
Delete(삭제)를 묶어서 일컫는 말

## API 문서

- 통신 규칙 = 표준 문서가 있어야 한다.
- 어떤 요청을 넣어야 하는지, 결과값은 어떤 것을 리턴하는지 등등

API정보

호출방식	요청 URL	출력결과
POST GET	https://business.juso.go.kr/addrlink/addrLinkApiJsonp.do (※ JAVA 1.6이상버전, TLS1.2 사용가능 이외 환경 http:// 사용)	JSONP(xml, json)
POST GET	https://business.juso.go.kr/addrlink/addrLinkApi.do (※ JAVA 1.6이상버전, TLS1.2 사용가능 이외 환경 http:// 사용)	xml, json

요청변수

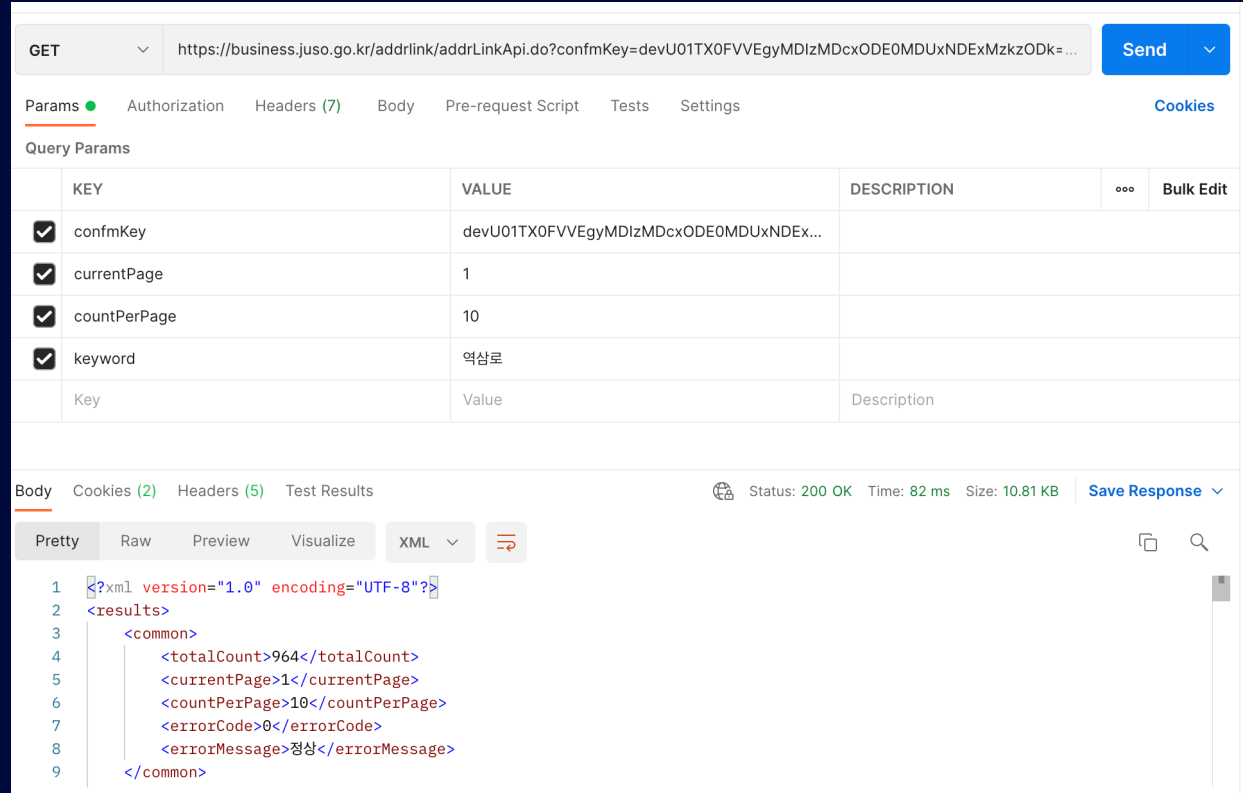
요청변수명	타입	필수여부	기본값	설명
confmKey	String	Y	-	신청시 발급받은 승인키
currentPage	Integer	Y	1	현재 페이지 번호
countPerPage	Integer	Y	10	페이지당 출력할 결과 Row 수
keyword	String	Y	-	주소 검색어
resultType	String	N	xml	검색결과형식 설정 (xml, json)
hstryYn	String	N	N	• 2020년12월8일 추가된 항목 변동된 주소정보 포함 여부
firstSort	String	N	none	• 2020년12월8일 추가된 항목 정확도순 정렬 (none), 우선정렬 (road: 도로명 포함, location: 지번 포함) ※ keyword(검색어)가 우선정렬 항목에 포함된 결과 우선 표출

-> Public API 는 다 이러한 API 문서가 공개되어 있다

# 3. API

## API 결과값 확인하기 - Postman

- Postman 은 API 결과값을 굉장히 쉽게 확인할 수 있게 해준다



The screenshot shows a Postman interface for a GET request to `https://business.juso.go.kr/addrlink/addrLinkApi.do?confmKey=devU01TX0FVVEgyMDIzMDcxODE0MDUxNDEzMzkzODk=...`. The 'Params' tab is active, displaying a table of query parameters:

KEY	VALUE	DESCRIPTION
confmKey	devU01TX0FVVEgyMDIzMDcxODE0MDUxNDE...	
currentPage	1	
countPerPage	10	
keyword	역삼로	
Key	Value	Description

The 'Body' tab is also active, showing the XML response in 'Pretty' format:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <results>
3   <common>
4     <totalCount>964</totalCount>
5     <currentPage>1</currentPage>
6     <countPerPage>10</countPerPage>
7     <errorCode>0</errorCode>
8     <errorMessage>정상</errorMessage>
9   </common>
```

At the top right of the interface, there is a 'Send' button. Below the 'Body' tab, there are buttons for 'Raw', 'Preview', 'Visualize', and 'XML'. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 82 ms', and 'Size: 10.81 KB'.

## python 으로 API 개발하기

```
url = "https://business.juso.go.kr/addrlink/addrLinkApi.do"

# Parameters
confmKey = "devU01TX0FVVEgyMDIzMDcxODE0MDUxNDEzMzkzODk="
currentPage = 1
countPerPage = 10
keyword = "서대문구 연세로 50"

# Construct the URL with encoded parameters
params = {
    "confmKey": confmKey,
    "currentPage": currentPage,
    "countPerPage": countPerPage,
    "keyword": keyword
}
encoded_params = "&".join([f"{key}={value}" for key, value in params.items()])
full_url = f"{url}?{encoded_params}"

# Send the GET request
response = requests.get(full_url)

# Process the response
if response.status_code == 200:
    xml_data = response.text
    data_dict = xmltodict.parse(xml_data)
    json_data = json.dumps(data_dict, ensure_ascii=False)
    parsed_data = json.loads(json_data)
    #logger.info(json_data)
    #logger.info(parsed_data)

    # Process the data as needed
else:
    # Request failed
    print(f"Request failed with status code {response.status_code}")
```

-> 실습해봅시다 !



## API 실습

(week1 의 api.py 참고)

1. Public API 찾아보기
2. Postman 으로 결과 확인
3. python 으로 코드 짜서 결과 확인해보기