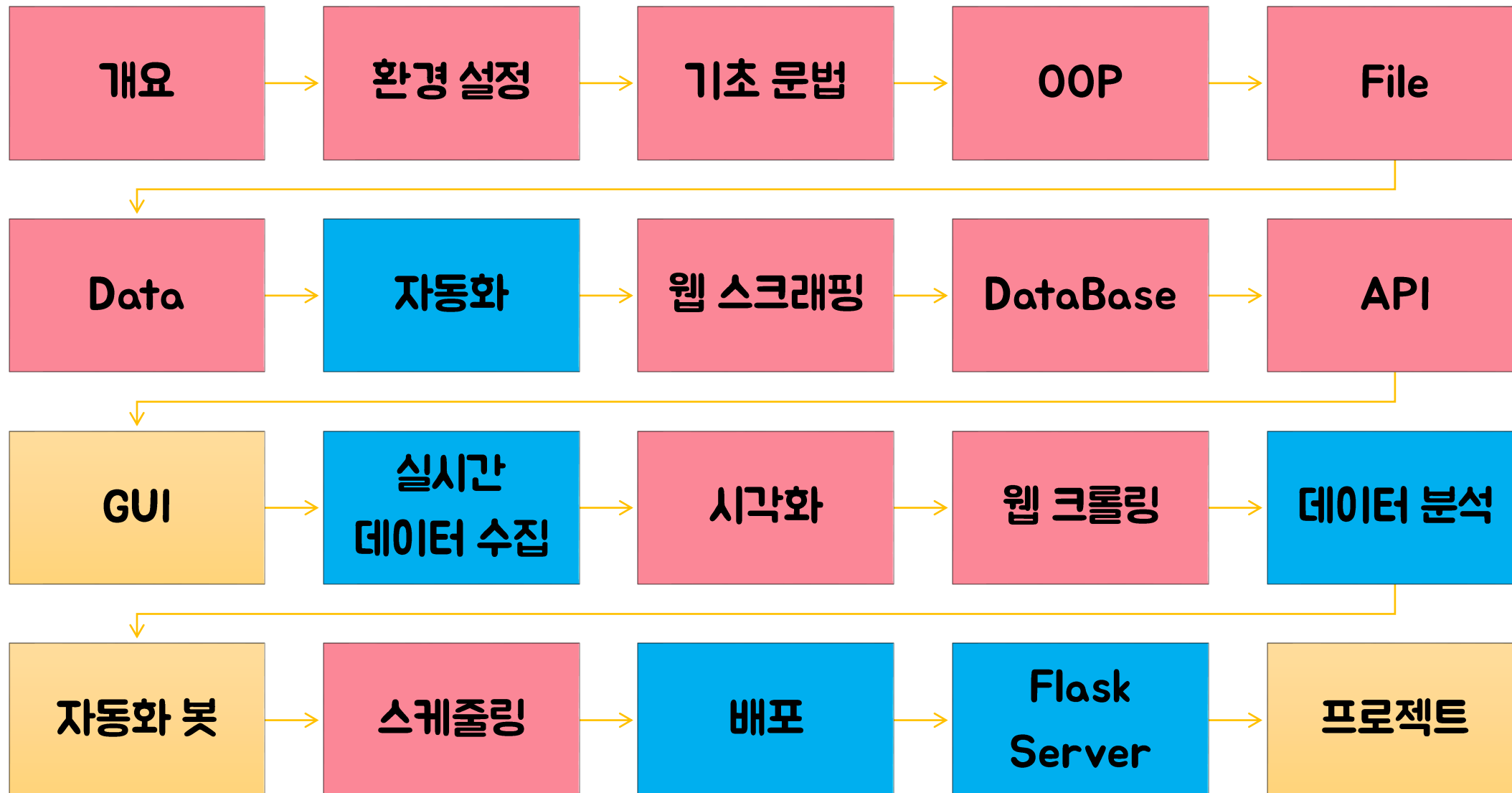




대한상공회의소
서울기술교육센터

나 예 호 교수



애플리케이션 간 데이터를 주고받을 수 있도록 해주는 인터페이스

`requests.get(url)` : API에 GET 요청을 보냄

`response.status_code` : 응답 코드 확인 (200 OK, 404 Not Found)

`response.json()` : JSON 데이터를 딕셔너리로 변환.

OpenWeather API를 활용하여 실시간 날씨 정보를 얻자

```
import requests
import json

# 1. OpenWeather API에서 날씨 정보 가져오기
API_KEY = "MY_API_KEY"
city = "Seoul"
url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"

response = requests.get(url)
data = response.json()
```

OpenWeather API를 활용하여 실시간 날씨 정보를 얻자

2. 온도 정보 출력

```
temperature = data["main"]["temp"]  
print(f"{city}의 현재 온도: {temperature}°C")
```

3. JSON 데이터 파일로 저장

```
with open("weather_data.json", "w") as file:  
    json.dump(data, file, indent=4)
```

OpenWeather API를 활용하여 실시간 날씨 정보를 얻자

4. API 요청 중 에러 발생 시 예외 처리

try:

response = requests.get(url)

response.raise_for_status()

except requests.exceptions.HTTPError **as** err:

print("HTTP 오류 발생:", err)

JSON Data를 활용해보자

```
import json
import pandas as pd

# 1. JSON 파일 불러오기
with open("weather_data.json", "r") as file:
    data = json.load(file)
```

```
# 2. 특정 키 값 출력 (온도 정보)
print("온도:", data["main"]["temp"])
```

JSON Data를 활용해보자

3. pandas를 활용한 데이터 변환

```
df = pd.DataFrame([data["main"]]) # 온도 관련 데이터만 추출  
df.to_csv("weather_data.csv", index=False)
```

4. JSON 데이터를 특정 키값으로 가공

```
weather_info = {  
    "city": data["name"],  
    "temperature": data["main"]["temp"],  
    "humidity": data["main"]["humidity"]  
}  
print(weather_info)
```

5. 엑셀 파일로 저장

```
df.to_excel("weather_data.xlsx", index=False)
```


JSON Data를 활용해보자

3. pandas를 활용한 데이터 변환

```
df = pd.DataFrame([data["main"]]) # 온도 관련 데이터만 추출  
df.to_csv("weather_data.csv", index=False)
```

4. JSON 데이터를 특정 키값으로 가공

```
weather_info = {  
    "city": data["name"],  
    "temperature": data["main"]["temp"],  
    "humidity": data["main"]["humidity"]  
}  
print(weather_info)
```

5. 엑셀 파일로 저장

```
df.to_excel("weather_data.xlsx", index=False)
```

사람이 관여하는 부분을 최소화

1. 파일 및 데이터

- 파일 입출력(open(), write(), read())
- CSV, JSON 파일 자동 처리(csv, json 모듈 활용)
- 파일 관리(os, shutil, glob 모듈 활용)

2. 웹 스크래핑 및 크롤링

- requests, BeautifulSoup을 활용한 데이터 자동 수집
- HTML 구조 분석 후 특정 데이터 분석
- 수집한 데이터를 CSV, JSON 등으로 저장
- Selenium을 활용한 동적 웹 페이지 자동 크롤링

사람이 관여하는 부분을 최소화

3. 데이터베이스 활용

- 데이터 삽입, 조회, 수정, 삭제 자동화(sqlite3 모듈 활용)
- SQL 쿼리를 활용한 자동 데이터 분석

4. API 활용 및 자동화

- API 데이터를 자동 요청 및 저장 (requests 활용)
 - * 영화진흥위원회 API, OpenWeather API
- JSON 데이터 자동 가공 및 엑셀 변환

사람이 관여하는 부분을 최소화

5. GUI 자동화

- 마우스 클릭 및 키보드 입력 자동화(PyAutoGUI 모듈 활용)
- 특정 버튼을 자동으로 클릭하는 프로그램 제작
- 이미지 인식을 활용한 UI 자동 조작

6. 업무 자동화

- 주기적인 작업 자동 실행(schedule 모듈 활용)
- 반복 실행 자동화(cron, Window Task scheduler 모듈 활용)
- 실행 파일 자동 배포(PyInstaller 모듈 활용)
- CI/CD 자동 배포 시스템 구축(Github Actions 모듈 활용)

PyAutoGUI란?

- 마우스, 키보드 입력을 자동화하는 라이브러리.
- 특정 작업을 자동화하거나 반복적인 업무를 줄이는 데 유용.

마우스 자동화 주요 함수

함수	설명
<code>pyautogui.moveTo(x, y, duration)</code>	마우스를 (x, y) 좌표로 이동
<code>pyautogui.click(x, y, clicks, interval)</code>	특정 좌표에서 클릭
<code>pyautogui.rightClick(x, y)</code>	우클릭
<code>pyautogui.doubleClick(x, y)</code>	더블 클릭
<code>pyautogui.dragTo(x, y, duration)</code>	마우스를 드래그

1. 마우스를 특정 좌표로 이동
2. 특정 위치에서 마우스를 클릭
3. 마우스를 2초 동안 천천히 이동한 후 클릭
4. 더블 클릭을 수행
5. 마우스를 드래그하여 특정 위치까지 이동

```
import pyautogui  
import time
```

1. 특정 좌표로 이동

```
pyautogui.moveTo(100, 200)
```

2. 특정 위치에서 클릭

```
pyautogui.click(300, 400)
```

3. 2초 동안 이동 후 클릭

```
pyautogui.moveTo(500, 500, duration=2)
```

```
pyautogui.click()
```

4. 더블 클릭 실행

```
pyautogui.doubleClick(600, 600)
```

5. 마우스를 드래그하여 이동

```
pyautogui.moveTo(700, 700)
```

```
pyautogui.dragTo(900, 900, duration=2)
```

PyAutoGUI란?

- 마우스, 키보드 입력을 자동화하는 라이브러리.
- 특정 작업을 자동화하거나 반복적인 업무를 줄이는 데 유용.

키보드 자동화 주요 함수

함수	설명
<code>pyautogui.write("hello")</code>	키보드 입력
<code>pyautogui.press("enter")</code>	특정 키 누르기
<code>pyautogui.hotkey("ctrl", "c")</code>	여러 키 조합 입력
<code>pyautogui.typewrite(["h", "e", "l", "l", "o"], interval=0.1)</code>	한 글자씩 입력

1. 키보드를 사용해 "Hello, World!"를 입력
2. Enter 키를 누르는 코드를 작성
3. "Ctrl + C"를 실행하는 프로그램을 작성
4. 특정 프로그램(예: 메모장)을 열고 "자동 입력 테스트"를 입력
5. "h", "e", "l", "l", "o"를 한 글자씩 입력하고, Enter 키를 누르기

```
import pyautogui  
import time
```

```
# 1. "Hello, World!" 입력  
time.sleep(2)  
pyautogui.write("Hello, World!", interval=0.1)
```

```
# 2. Enter 키 누르기  
pyautogui.press("enter")  
  
# 3. "Ctrl + C" 실행  
pyautogui.hotkey("ctrl", "c")
```

4. 메모장 실행 후 "자동 입력 테스트" 입력

```
pyautogui.hotkey("win", "r") # 실행 창 열기
```

```
time.sleep(1)
```

```
pyautogui.write("notepad")
```

```
pyautogui.press("enter")
```

```
time.sleep(1)
```

```
pyautogui.write("자동 입력 테스트", interval=0.1)
```

5. 한 글자씩 입력 후 Enter

```
pyautogui.write(["h", "e", "\\", "\\", "o"], interval=0.1)
```

```
pyautogui.press("enter")
```

이미지 인식을 활용한 자동화 (윈도우 확인 필요)

`pyautogui.locateOnScreen(image_path)`

- 특정 이미지가 화면에 있는지 검색

`pyautogui.locateCenterOnScreen(image_path)`

- 특정 이미지의 중앙 좌표 찾기

1. 메모장을 실행하고, "자동화 실행 중" 입력
2. 저장 버튼 클릭 후 파일명 "auto_test.txt" 입력
3. "확인" 버튼을 클릭하여 저장 완료

```
import pyautogui
import time

# 1. 메모장 실행
pyautogui.hotkey("win", "r") # 실행 창 열기
time.sleep(1)
pyautogui.write("notepad")
pyautogui.press("enter")
time.sleep(2)
```

2. "자동화 실행 중" 입력

```
pyautogui.write("자동화 실행 중", interval=0.1)
```

3. "Ctrl + S"로 저장 창 열기

```
pyautogui.hotkey("ctrl", "s")
```

```
time.sleep(2)
```

4. 파일명 입력

```
pyautogui.write("auto_test.txt")
```

```
time.sleep(1)
```

```
pyautogui.press("enter") # 저장
```

5. 저장 확인 버튼 클릭 (이미지 인식 활용)

```
confirm_btn = pyautogui.locateCenterOnScreen("confirm.png")
if confirm_btn:
    pyautogui.click(confirm_btn)

print("자동화 완료!")
```


특정 프로그램을 자동으로 일정한 시간마다 실행하는 기능 반복적인 작업(백업, 데이터 수집)을 자동화할 때 사용

방법	설명	플랫폼
schedule 라이브러리	Python에서 스케줄링을 쉽게 구현	Windows, Mac, Linux
cron	리눅스에서 가장 많이 사용되는 스케줄러	Linux, Mac
Windows Task Scheduler	윈도우에서 제공하는 자동화 도구	Windows

```
!pip3 install schedule
```

```
import schedule
import time

def job_2():
    print("2초마다 실행되는 작업!")
```

```
# 2초마다 실행
task2 = schedule.every(2).seconds.do(job_2)
```

```
try:
    while True:
        schedule.run_pending()
        time.sleep(1)
finally:
    schedule.cancel_job(task2)
    print("스케줄링 종료")
```

```
while True:
    schedule.run_pending()
    time.sleep(60)  # 1분마다 실행 체크
```

```
try:
    while True:
        schedule.run_pending()
        time.sleep(1)
finally:
    schedule.cancel_job(task2)
    print("스케줄링 종료")
```

```
finally:
    # 모든 작업 정지
    schedule.clear()
```

```
# 3초마다 실행
task3 = schedule.every(3).seconds.do(lambda : print("3초마다 실행"))
```

```
start_time = time.time() # 시작 시간 기록

while True:
    schedule.run_pending()
    time.sleep(1)

# 30초가 지나면 스케줄 종료
if time.time() - start_time > 30:
    print("30초 후 스케줄 종료!")
    schedule.clear()
    break
```

1. 날씨 데이터 요청 함수

```
def fetch_weather():  
    response = requests.get(url)  
    data = response.json()  
    df = pd.DataFrame([data["main"]])  
    df.to_csv("daily_weather_report.csv", index=False)  
    print(f"{city} 날씨 데이터 저장 완료!")
```

2. 일정 주기로 실행

```
schedule.every().day.at("08:00").do(fetch_weather)
```

```
while True:
```

```
    schedule.run_pending()
```

```
    time.sleep(60) # 1분마다 실행 체크
```

Python 코드를 .exe 파일로 변환
Python이 설치되지 않은 환경에서도 실행 가능
배포가 편리해지고, 일반 사용자가 쉽게 실행할 수 있음.

```
!pip install pyinstaller
```

```
pyinstaller --onefile test.py
```

GitHub

- 코드 버전 관리 시스템(Git)을 온라인에서 공유하는 플랫폼
- 협업 및 자동화 배포(CI/CD) 지원

CI/CD

- CI(Continuous Integration, 지속적 통합)
 - : 코드 변경 사항을 자동으로 테스트 및 빌드
- CD(Continuous Deployment, 지속적 배포)
 - : 자동으로 애플리케이션을 배포

GitHub Actions을 활용한 자동 배포

- 특정 이벤트(push, pull request 등)가 발생했을 때 자동으로 실행되는 GitHub의 CI/CD 기능.