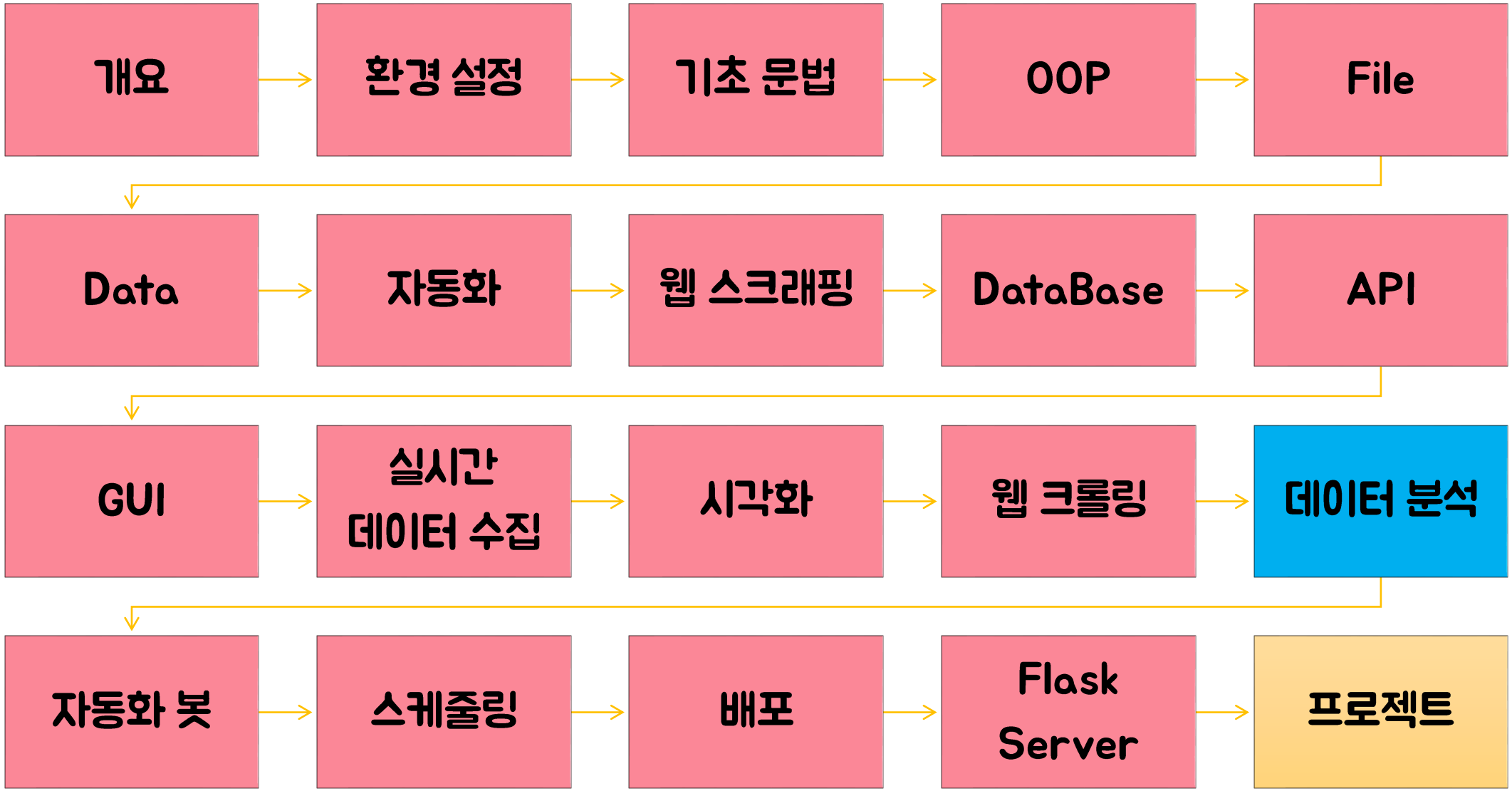
python

대한상공회의소
서울기술교육센터

나 예 호 교수

python 수업 진행

대한상공회의소
서울기술교육센터



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
pyautogui.moveTo(x, y, duration)	마우스를 (x, y) 좌표로 이동
pyautogui.click(x, y, clicks, interval)	특정 좌표에서 클릭
pyautogui.rightClick(x, y)	우클릭
pyautogui.doubleClick(x, y)	더블 클릭
pyautogui.dragTo(x, y, duration)	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
```

```python
# 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
pyautogui.write("hello")	키보드 입력
pyautogui.press("enter")	특정 키 누르기
pyautogui.hotkey("ctrl", "c")	여러 키 조합 입력
pyautogui.typewrite(["h", "e", "l", "l", "o"], interval=0.1)	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", "l", "l", "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

```python
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
```

```python
# 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
```

python API + 스케줄링


```python
# 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!")
```

```python
# 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.

.yml (Yaml Ain't Markup Language)

- GitHub Actions의 자동화된 작업을 설정하기 위해 사용해보자

yml이란
기존에 사용되었던 속성 구조

```
[properties]
weather_seoul.server.url=127.0.0.1
weather_seoul.server.port=5000
weather_seoul.server.user=user
weather_seoul.server.password=password
```

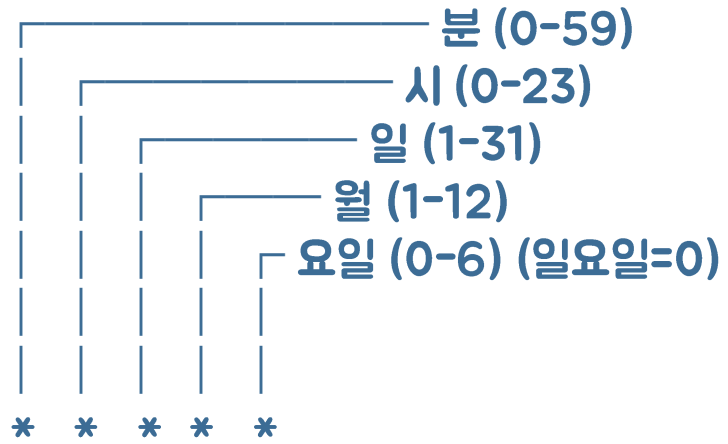
yml을 활용한 계층 속성 구조

```
[properties]
weather_seoul:
    server:
        url:127.0.0.1
        port:5000
        user:user
        password:password
```

.yml을 구성해보자

```
name: Fetch Seoul Weather Data # 스크립트 이름

on: # 작업을 수행하는 조건
  schedule: # 스케줄링
    - cron: "*/1 * * * *"  # 1분마다 실행
  workflow_dispatch:  # 수동 실행 가능
  push:
    branches:
      - main # 메인에 push발생 시 작업 수행
```



cron 설정	실행 주기	한국 시각(KST) (UTC+9)
*/1 * * * *	1분 마다	1분 마다
*/5 * * * *	5분 마다	5분 마다
0 * * * *	매 정각 (매시 0분)	매 정각
30 2 * * *	매일 UTC 02:30	매일 KST 11:30
15 14 * * *	매일 UTC 14:15	매일 KST 23:15
0 9 * * 1	매주 월요일 UTC 09:00	매주 월요일 KST 18:00
0 0 1 * *	매월 1일 UTC 00:00	매월 1일 KST 09:00

.yml을 구성해보자

```
jobs: # 실행할 작업 정의
  fetch_weather: #작업 이름
    runs-on: ubuntu-latest # 이 스크립트는 우분투 최신환경에서 실행
                           # GitHub Actions에서 제공하는 가상 머신(런너) 중 하나
    steps: # 작업 단계 시작          # windows-latest와 macos-latest도 가능 그러나
      - name: 저장소 체크아웃 # 각 - name: 단계는 GitHub Actions에서 실행될 개별 작업
        uses: actions/checkout@v3 # 현재 GitHub 저장소를 가져옴 (clone)
        with: # 추가적인 설정을 전달하는 역할 (옵션)
          token: ${{ secrets.GITHUB_TOKEN }}  # 푸시 권한 추가


      - name: Python 설정
        uses: actions/setup-python@v4 # Python을 설치하는 GitHub Actions 플러그인
        with:
          python-version: "3.9 "  # Python 3.9 버전을 사용
```

.yml을 구성해보자

```
- name: 필요한 패키지 설치
  run: pip install requests # API 요청을 보낼 때 필요

- name: 날씨 데이터 가져오기
  env: # GitHub Secrets에서 환경 변수를 가져옴
    OPENWEATHER_API_KEY: ${{ secrets.OPENWEATHER_API_KEY }}
  run: python weather_script.py # Python 스크립트를 실행

- name: 변경 사항 커밋 및 푸시
  run: |
    git config --global user.name "github-actions[bot]"  # 자동 Push시 사용하는 기본 계정
    git config --global user.email "github-actions@github.com" #개인 이메일 권장 X
    git add seoul_weather.csv # 날씨 데이터를 저장한 CSV 파일을 GitHub에 추가
    git commit -m "Update weather data (auto)" # 변경 사항이 있을 경우 커밋
    git push # 변경 사항을 git에 push
```