SK쉴더스 클라우드기반 스마트 융합보안 과정 26기(AI)

PBL 1번

모범 답안

```python
class StudentScores:
    def __init__(self, filename):
        """파일에서 데이터를 읽어 scores 속성에 저장합니다."""
        self.scores = {}  # 학생 이름과 점수를 저장할 딕셔너리
        try:
            # 주어진 파일을 읽어서 데이터를 처리
            with open(filename, "r", encoding="utf-8") as file:
                for line in file:
                    # 각 줄을 이름과 점수로 분리하고 딕셔너리에 저장
                    name, score = line.strip().split(",")
                    self.scores[name] = int(score)  # 점수는 정수로 변환
        except FileNotFoundError:
            # 파일이 없을 경우 오류 메시지 출력
            print(f"{filename} 파일이 존재하지 않습니다.")
        except Exception as e:
            # 다른 예외 발생 시 오류 메시지 출력
            print(f"오류가 발생했습니다: {e}")

    def calculate_average(self):
        """평균 점수를 계산하여 반환합니다."""
        total = sum(self.scores.values())  # 점수의 총합 계산
        return total / len(self.scores)  # 평균 계산

    def get_above_average(self):
        """평균 점수 이상을 받은 학생들의 이름 리스트를 반환합니다."""
        average = self.calculate_average()  # 평균 점수 계산
        # 평균 점수 이상인 학생들의 이름 리스트 생성
        return [name for name, score in self.scores.items() if score >= average]

    def save_below_average(self, output_filename):
        """평균 이하 점수를 받은 학생들의 데이터를 파일로 저장합니다."""
        average = self.calculate_average()  # 평균 점수 계산
        with open(output_filename, "w", encoding="utf-8") as file:
            # 평균 이하 학생들의 데이터를 파일에 작성
            for name, score in self.scores.items():
                if score < average:
                    file.write(f"{name},{score}\n")

    def print_summary(self):
```

```
        """평균 점수와 평균 이상 학생 리스트를 출력합니다."""
        average = self.calculate_average()  # 평균 점수 계산
        above_average = self.get_above_average()  # 평균 이상 학생 리스트 생성
        # 결과 출력
        print(f"평균 점수: {average:.1f}")
        print(f"평균 이상을 받은 학생들: {above_average}")


# 프로그램 실행
filename = "scores_korean.txt"  # 한글 이름이 포함된 입력 파일
output_filename = "below_average_korean.txt"  # 평균 이하 학생 데이터를 저장할 출력 파일

# StudentScores 객체 생성 및 처리
student_scores = StudentScores(filename)  # 파일에서 데이터를 읽어 객체 생성
student_scores.print_summary()  # 평균 점수와 평균 이상 학생 리스트 출력
student_scores.save_below_average(output_filename)  # 평균 이하 학생 데이터를 파일에 저장
```

PBL 2번

모범 답안

```
import re
from collections import Counter
import csv
import os

# IP 추출 함수
# 로그 파일에서 IP 주소를 추출하는 함수
def extract_ips_from_log(file_path):
    try:
        # 로그 파일을 읽기 모드로 열기
        with open(file_path, "r") as log_file:
            log_data = log_file.read()  # 로그 파일 전체 내용을 읽어들임

        # 정규 표현식을 사용하여 IP 주소 추출
        # IP 주소 형식: 숫자.숫자.숫자.숫자 (0~255 범위, 여기서는 단순한 형식만 검출)
        ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'
        # 정규 표현식에 매칭되는 IP 주소를 모두 찾음
        ip_addresses = re.findall(ip_pattern, log_data)
        return ip_addresses  # 추출된 IP 주소 리스트 반환
    except FileNotFoundError:  # 파일이 존재하지 않을 경우
        print("로그 파일을 찾을 수 없습니다.")  # 에러 메시지 출력
        return None  # None 반환

# IP 빈도 분석 함수
# 추출된 IP 주소들의 빈도를 계산하는 함수
def analyze_ip_count(ip_addresses):
    return Counter(ip_addresses)  # IP 주소 리스트를 입력받아 각 IP의 개수를 세는 Counter
객체 반환

# CSV 저장 함수
# 분석 결과를 CSV 파일로 저장하는 함수
def save_to_csv(counter, output_file):
    # CSV 파일을 한글이 깨지지 않도록 utf-8-sig 인코딩으로 저장
    with open(output_file, mode='w', newline='', encoding='utf-8-sig') as file:
        writer = csv.writer(file)  # CSV 작성 객체 생성
        writer.writerow(["IP 주소", "접속 횟수"])  # 헤더 작성
        # Counter 객체의 아이템(IP, 빈도)을 행 단위로 작성
        writer.writerows(counter.items())
    # 저장 완료 메시지 출력
```

```
    print(f"분석 결과가 '{output_file}' 파일에 저장되었습니다.")

# 실행
log_file_path = "sample_log_file.log"  # 분석할 로그 파일의 경로
output_file = "ip_analysis.csv"  # 분석 결과를 저장할 CSV 파일의 경로

# 로그 파일 존재 여부 확인
if not os.path.exists(log_file_path):  # 로그 파일이 존재하지 않으면
    # 오류 메시지 출력
    print(f"로그 파일 '{log_file_path}'이(가) 존재하지 않습니다.")
    exit(1)  # 프로그램 종료

# IP 주소 추출
ips = extract_ips_from_log(log_file_path)
if ips:  # IP 주소가 정상적으로 추출되었을 경우
    # IP 빈도 분석
    ip_count = analyze_ip_count(ips)

    # 상위 3개 IP 주소와 빈도를 출력
    print("\nIP 주소 빈도 분석 결과 (상위 3개):")
    # 가장 많이 등장한 상위 3개 IP 출력
    for ip, count in ip_count.most_common(3):
        print(f"{ip}: {count}회")

    # 분석 결과를 CSV 파일로 저장
    save_to_csv(ip_count, output_file)
```

주요 설명

- 로그 파일에서 IP 추출 (extract_ips_from_log)
  → 정규 표현식(re)을 사용해 IP 주소 형식에 맞는 데이터를 추출합니다.
  정규식 r'₩b(?:[0-9]{1,3}₩.){3}[0-9]{1,3}₩b'은 IPv4 주소 형식을 검출합니다.
- IP 빈도 계산 (analyze_ip_count)
  → collections.Counter를 사용해 추출된 IP 주소들의 개수를 계산합니다.
- CSV 파일 저장 (save_to_csv)
  → 분석된 결과를 CSV 형식으로 저장하며, 인코딩을 utf-8-sig로 설정하여 한글 데이터를 정상적으로 처리합니다.
- 상위 3개 IP 출력
  → Counter.most_common(n) 메서드를 사용해 빈도가 높은 상위 3개의 IP 주소를 출력합니다.
- 오류 처리
  → **로그 파일이 없을 경우 파일이 없다는 메시지를 출력하고 프로그램을 종료합니다.**

PBL 3번

모범 답안

```
import os
import time
import re

# 모니터링할 디렉터리 경로
MONITOR_DIR = "./monitor_directory"

# 감지할 주요 정보 패턴 정의
# PATTERNS 딕셔너리의 키는 탐지하려는 정보 유형이며, 값은 해당 정보를 찾기 위한 정규
표현식입니다.
PATTERNS = {
    "comments": r"#.*",  # 주석: '#'로 시작하는 모든 문자열
    "email": r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}",  # Email 주소: 일반적인
이메일 형식
    "sql": r"SELECT|INSERT|UPDATE|DELETE|DROP|CREATE|ALTER"  # SQL 키워드: 주요 SQL 명령어
}

# 주의 대상 확장자: 모니터링 대상 파일 중 주의가 필요한 확장자 목록
WATCH_EXTENSIONS = (".js", ".class", ".py")

# 이미 존재하는 파일 리스트를 초기화
def get_initial_files(directory):
    """
    주어진 디렉터리의 파일 목록을 반환.
    디렉터리가 존재하지 않을 경우 빈 집합을 반환.
    Args: directory (str): 모니터링할 디렉터리 경로.
    Returns: set -> 디렉터리에 존재하는 파일 이름들의 집합.
    """
    return set(os.listdir(directory)) if os.path.exists(directory) else set()

# 파일 내용에서 주요 정보 검색
def scan_file_for_issues(filepath):
    """
    파일 내용을 확인하여 주요 정보(주석, 이메일, SQL 코드 등)를 탐지.
    Args: filepath (str): 파일 경로.
    Returns: list-> 발견된 주요 정보의 목록 (줄 번호, 내용, 유형).
    """
    issues = []
    with open(filepath, "r", encoding="utf-8") as file:
```

```
        for line_number, line in enumerate(file, start=1):  # 파일을 한 줄씩 읽고 줄 번호를
함께 반환
            for issue_type, pattern in PATTERNS.items():  # 주요 정보 패턴과 매칭
                if re.search(pattern, line):  # 패턴과 일치하는 내용이 있으면
                    issues.append((line_number, line.strip(), issue_type))  # 줄 번호,
내용, 유형 저장
    return issues

# 디렉터리를 모니터링하고 새로운 파일을 분석
def monitor_directory(directory, known_files):
    """
    디렉터리를 모니터링하여 새로운 파일을 감지하고 분석.
    Args: directory (str)-> 모니터링할 디렉터리 경로.
        known_files (set)-> 이미 존재하는 파일 이름들의 집합.
    Returns:
        set: 업데이트된 파일 이름들의 집합.
    """
    # 현재 디렉터리에 존재하는 파일 목록
    current_files = set(os.listdir(directory))
    # 새로운 파일 감지
    new_files = current_files - known_files

    if new_files:  # 새로운 파일이 발견된 경우
        print(f"[INFO] 새로운 파일이 추가되었습니다: {new_files}")

        for new_file in new_files:
            filepath = os.path.join(directory, new_file)  # 파일의 전체 경로 생성

            # 파일 확장자가 주의 대상인 경우 경고 출력
            if new_file.endswith(WATCH_EXTENSIONS):
                print(f"[WARNING] 주의 파일 발견: {new_file}")

            # 파일 내용 스캔하여 주요 정보 탐지
            try:
                issues = scan_file_for_issues(filepath)
                if issues:  # 주요 정보가 발견된 경우
                    print(f"[ALERT] {new_file}에서 중요한 정보가 발견되었습니다:")
                    for issue in issues:
                        line_number, content, issue_type = issue
                        print(f"  - {issue_type} (줄 {line_number}): {content}")
            except Exception as e:  # 파일 분석 중 오류 발생 시
                print(f"[ERROR] {new_file} 분석 중 오류 발생: {e}")

    return current_files  # 업데이트된 파일 목록 반환

if __name__ == "__main__":
    # 모니터링할 디렉터리가 존재하는지 확인
```

```
    if not os.path.exists(MONITOR_DIR):
        print(f"[ERROR] 디렉터리 {MONITOR_DIR}가 존재하지 않습니다.")
        exit(1)  # 프로그램 종료

    print(f"[INFO] {MONITOR_DIR} 디렉터리 모니터링 시작...")
    # 초기 파일 리스트 가져오기
    known_files = get_initial_files(MONITOR_DIR)

    try:
        while True:
            # 디렉터리 모니터링 및 업데이트
            known_files = monitor_directory(MONITOR_DIR, known_files)
            time.sleep(1)  # 1초 간격으로 디렉터리 확인
    except KeyboardInterrupt:  # 사용자가 Ctrl+C를 눌러 종료할 경우
        print("[INFO] 모니터링 종료")
```

주요 설명

- **초기 파일 확인**

  get_initial_files 함수로 모니터링 시작 시 존재하는 파일 목록을 기록합니다.

- **새 파일 감지**

  디렉터리 내 새롭게 생성된 파일만을 필터링하여 출력합니다.

- **특정 확장자 경고**

  새로 생성된 파일 중 특정 확장자를 가진 파일이 발견되면 경고를 출력합니다.

- **파일 내용 스캔**

  새 파일 내용을 읽어 주석, 이메일, SQL 코드 등 주요 정보를 패턴 매칭으로 검색하고, 발견된 줄 번호와 내용을 출력합니다.

- **주기적 모니터링**

  무한 루프에서 10초 간격으로 디렉터리를 확인하며, 키보드 인터럽트 시 종료됩니다.

PBL 4번

모범 답안

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

class SalesAnalysis:
    def __init__(self):
        """매출 데이터 초기화"""
        np.random.seed(42)
        dates = pd.date_range('2024-01-01', '2024-12-31', freq='D')
        sales = np.random.randint(1000, 10000, len(dates))
        self.__df = pd.DataFrame({'날짜': dates, '매출': sales})

    def visualize_monthly_sales(self):
        """월별 매출 총합을 시각화"""
        self.__df['월'] = self.__df['날짜'].dt.month
        monthly_sales = self.__df.groupby('월')['매출'].sum()
        # 한글 폰트 설정
        plt.rcParams['font.family'] = 'NanumGothic'
        plt.figure(figsize=(8, 5))
        plt.plot(monthly_sales.index, monthly_sales.values, marker='o')
        plt.title('월별 매출 총합')
        plt.xlabel('월')
        plt.ylabel('매출')
        plt.show()

# 클래스 인스턴스 생성
analysis = SalesAnalysis()
analysis.visualize_monthly_sales()
```

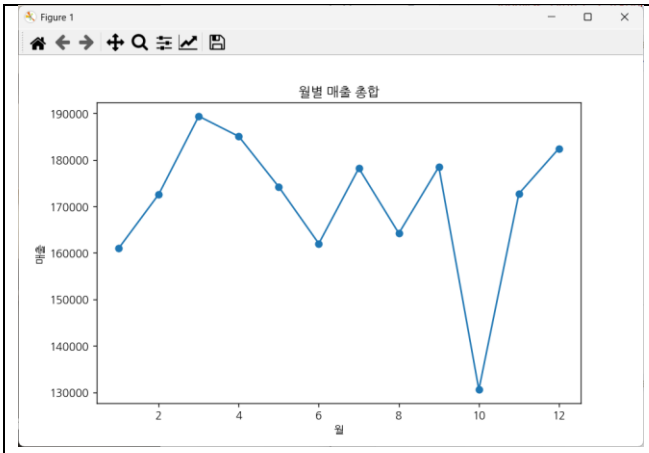


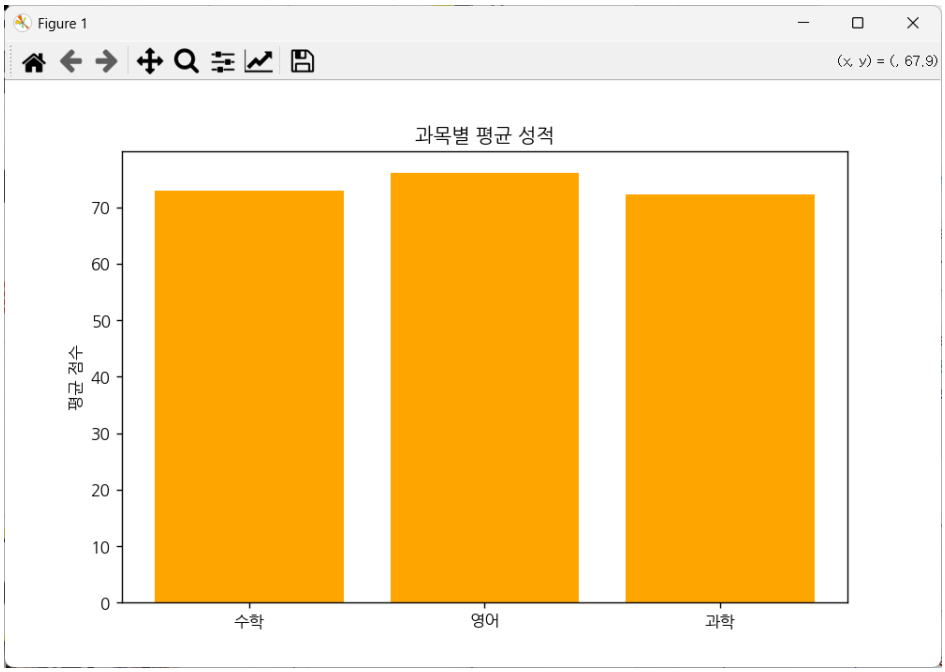
Figure 1
월별 매출 총합
190000
180000
170000
160000
150000
140000
130000
매출
2
4
6
8
10
12
월

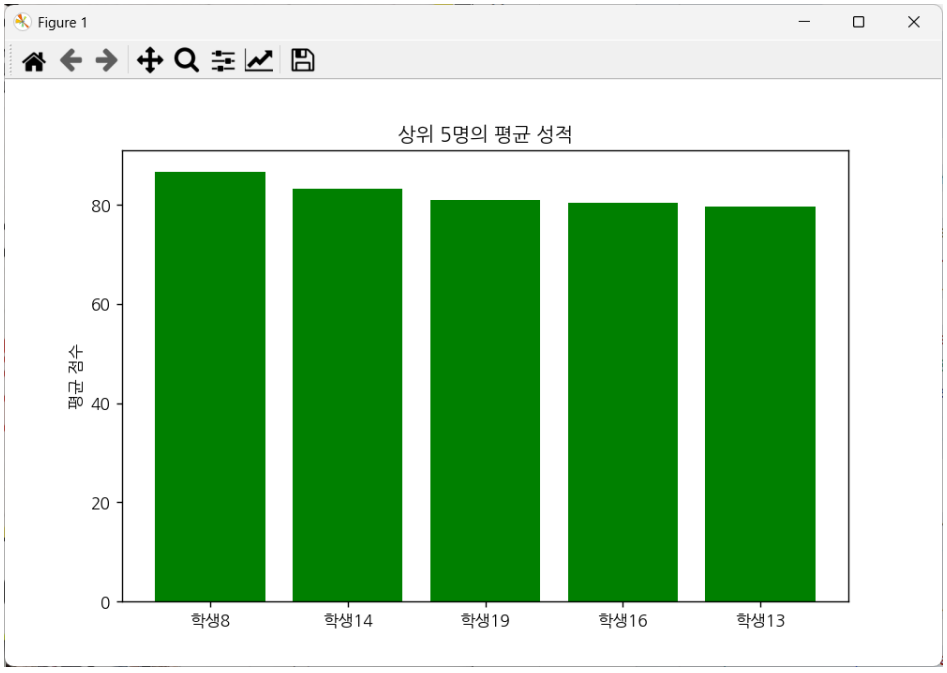
PBL 5번

모범 답안

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

class StudentScoreAnalysis:
    def __init__(self):
        """학생 성적 데이터 초기화"""
        np.random.seed(42)
        data = {
            '이름': ['학생' + str(i) for i in range(1, 21)],
            '수학': np.random.randint(50, 100, 20),
            '영어': np.random.randint(50, 100, 20),
            '과학': np.random.randint(50, 100, 20)
        }
        self.__df = pd.DataFrame(data)

    def visualize_subject_means(self):
        """과목별 평균 성적을 시각화"""
        subject_means = self.__df[['수학', '영어', '과학']].mean()
        # 한글 폰트 설정
        plt.rcParams['font.family'] = 'NanumGothic'
        plt.figure(figsize=(8, 5))
        plt.bar(subject_means.index, subject_means.values, color='orange')
        plt.title('과목별 평균 성적')
        plt.ylabel('평균 점수')
        plt.show()

    def visualize_top5_students(self):
        """평균 성적 상위 5명을 시각화"""
        self.__df['평균'] = self.__df[['수학', '영어', '과학']].mean(axis=1)
        top5_students = self.__df.sort_values(by='평균', ascending=False).head(5)
        # 한글 폰트 설정
        plt.rcParams['font.family'] = 'NanumGothic'
        plt.figure(figsize=(8, 5))
        plt.bar(top5_students['이름'], top5_students['평균'], color='green')
        plt.title('상위 5명의 평균 성적')
        plt.ylabel('평균 점수')
        plt.show()
```

```python
# 클래스 인스턴스 생성
analysis = StudentScoreAnalysis()
analysis.visualize_subject_means()
analysis.visualize_top5_students()
```





PBL 6번

모범 답안

```
import pandas as pd
import matplotlib.pyplot as plt

class CustomerSalesAnalysis:
    def __init__(self, data):
        """고객 매출 데이터 초기화"""
        self.__df = pd.DataFrame(data)
        self.__df['구매일자'] = pd.to_datetime(self.__df['구매일자'])
        self.__df['총매출'] = self.__df['수량'] * self.__df['단가']

    def visualize_monthly_sales(self):
        """월별 매출 총합을 막대 그래프로 시각화"""
        monthly_sales = self.__df.groupby(self.__df['구매일자'].dt.month)['총매출'].sum()
        # 한글 폰트 설정
        plt.rcParams['font.family'] = 'NanumGothic'
        plt.figure(figsize=(8, 5))
        plt.bar(monthly_sales.index, monthly_sales.values, color='skyblue')
        plt.title('월별 매출 총합')
        plt.xlabel('월')
        plt.ylabel('매출')
        plt.show()

    def visualize_customer_sales(self):
        """고객별 누적 매출을 파이 차트로 시각화"""
        customer_sales = self.__df.groupby('고객명')['총매출'].sum()
        plt.figure(figsize=(8, 8))
        plt.pie(customer_sales, labels=customer_sales.index, autopct='%1.1f%%',
startangle=140)
        plt.title('고객별 누적 매출 비율')
        plt.show()

if __name__ == '__main__':
    # 데이터 초기화
    data = {
        '고객명': ['홍길동', '이영희', '김철수', '박지수', '최민호', '홍길동', '이영희',
'김철수'],
        '구매일자': ['2024-01-10', '2024-02-14', '2024-02-18', '2024-03-05', '2024-03-
20', '2024-04-10', '2024-04-25', '2024-05-05'],
```

```
        '상품명': ['노트북', '키보드', '모니터', '노트북', '마우스', '모니터', '노트북',
'키보드'],
        '수량': [1, 2, 1, 2, 4, 2, 1, 3],
        '단가': [1500000, 50000, 300000, 1500000, 20000, 300000, 1500000, 50000]
    }
    # 클래스 인스턴스 생성
    analysis = CustomerSalesAnalysis(data)
    analysis.visualize_monthly_sales()
    analysis.visualize_customer_sales()
```