
(보안데이터분석) 연습문제_12

1. 파이썬을 사용하여 간단한 모델 서비스의 파일럿 구현에 적합한 도구인 Streamlit에 대한 설명으로 옳은 것은?
 1. Streamlit은 주로 백엔드 API를 구축하고 데이터베이스와 상호작용하는데 사용되는 프레임워크이다.
 2. Streamlit은 복잡한 JavaScript 프레임워크 지식 없이도 인터랙티브한 웹 애플리케이션을 빠르게 개발할 수 있도록 돕는다. ✓
 3. Streamlit은 모델 학습 및 배포를 위한 강력한 분산 컴퓨팅 환경을 제공한다.
 4. Streamlit은 주로 대규모 프로덕션 환경에서 모델 서비스를 운영하기 위한 엔터프라이즈 솔루션으로 설계되었다.
 5. Streamlit은 데이터 시각화 라이브러리와는 독립적으로 작동하며, 자체적인 시각화 기능을 제공하지 않는다.
2. 파이썬의 기본 GUI(Graphical User Interface) 개발 라이브러리인 Tkinter에 관한 설명으로 옳은 것은?
 1. Tkinter는 파이썬의 표준 라이브러리에 포함되어 있지 않아 별도의 설치가 필요하다.
 2. Tkinter는 웹 기반 애플리케이션 개발에 주로 사용되며, 웹 프레임워크와 긴밀하게 통합되어 있다.
 3. Tkinter로 개발된 애플리케이션은 Windows, macOS, Linux 등 다양한 운영체제에서 동일한 코드베이스로 실행될 수 있다. ✓
 4. Tkinter는 고도로 복잡하고 시각적으로 화려한 3D 그래픽 애플리케이션 개발에 최적화되어 있다.
 5. Tkinter는 Python 2.x 버전까지만 지원하며, Python 3.x 버전에서는 더 이상 사용되지 않는다.
3. 파이썬 GUI 개발 라이브러리에 대한 설명으로 옳지 않은 것은?
 - 1) **PyQt/PySide**는 Qt 프레임워크를 기반으로 하며, 복잡하고 상업적인 애플리케이션 개발에 적합하다.
 - 2) **Kivy**는 주로 모바일 환경의 멀티터치 애플리케이션 개발에 강점을 가지며, 독자적인 KV Language를 사용하여 UI를 정의할 수 있다.
 - 3) **wxPython**은 운영체제의 네이티브 위젯을 활용하여 해당 OS의 기본 UI와 유사한 외관을 제공하는 특징이 있다.
 - 4) **PySimpleGUI**는 기존의 복잡한 GUI 라이브러리 위에 추상화 계층을 제공하여, 매우 적은 코드량으로 GUI를 빠르게 구축할 수 있도록 돕는다.
 - 5) **PyQt/PySide, Kivy, wxPython**은 모두 파이썬 표준 라이브러리에 포함되

어 있어 별도의 설치 과정 없이 바로 사용할 수 있다. ✓

- PyQt / PySide: Qt 프레임워크의 파이썬 바인딩입니다. 매우 강력하고 기능이 풍부하여 복잡하고 상업적인 애플리케이션 개발에 널리 사용됩니다. Qt Designer와 같은 시각적 도구를 지원하여 UI 디자인이 용이. PyQt는 GPL 라이선스를 따르며, PySide는 LGPL 라이선스를 따름.

- Kivy: 멀티터치 애플리케이션 개발에 특화된 오픈소스 프레임워크. 데스크톱뿐만 아니라 iOS, Android 등 모바일 플랫폼에서도 동작하며, 독자적인 그래픽 렌더링 엔진(OpenGL ES 2 기반)을 사용하여 유려한 UI를 구현할 수 있음. KV Language라는 선언적 언어를 사용하여 UI를 정의할 수 있음.
- wxPython: C++로 작성된 wxWidgets 라이브러리의 파이썬 바인딩. 운영체제의 네이티브 위젯을 사용하여 애플리케이션이 해당 OS의 기본 UI와 유사하게 보이도록 함. 크로스 플랫폼 지원이 뛰어나며, 비교적 경량.
- PySimpleGUI: PyQt, Tkinter, Kivy, Web 등 다양한 백엔드를 사용하여 간단하고 빠르게 GUI를 만들 수 있도록 돕는 상위 레벨의 래퍼(wrapper) 라이브러리. 코드량이 적고 배우기 쉬워 초보자에게 특히 유용.`

- 다음은 간단한 pyside 를 이용한 버튼 구현 코드이다.

```
import sys
import random
import sys from PySide6.QtWidgets
import QApplication, QWidget, QPushButton, QVBoxLayout, QLabel
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAff as
FigureCanvas from matplotlib.figure import Figure

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("PySide6 버튼 예제")
        self.setGeometry(100, 100, 300, 150)

        layout = QVBoxLayout()
        self.label = QLabel("버튼을 눌러주세요!", self)
        layout.addWidget(self.label)

        self.figure = Figure(figsize=(5, 4), dpi=100)
        self.canvas = FigureCanvas(self.figure)

        layout.addWidget(self.canvas)
        self.axes = self.figure.add_subplot(111)
        self.update_plot() # 초기 그래프 그리기

        self.button = QPushButton("그래프 업데이트", self)
        self.button.clicked.connect(self.on_button_clicked)
        layout.addWidget(self.button)

        self.setLayout(layout)
```

```

def on_button_clicked(self):
    self.label.setText("버튼이 클릭되었습니다!")
    print("그래프 업데이트!")
    self.update_plot()

def update_plot(self):
    self.axes.clear()
    x = [i for i in range(10)]
    y = [random.randint(0, 100) for _ in range(10)]

    self.axes.plot(x, y, marker='o', linestyle='--')
    self.axes.set_title("랜덤 데이터 그래프")
    self.axes.set_xlabel("X축")
    self.axes.set_ylabel("Y축")
    self.axes.grid(True)
    self.canvas.draw()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MyWindow()
    window.show()
    sys.exit(app.exec())

```

4. 위 PySide6 코드 예시에서 버튼 클릭 시 특정 동작을 수행하도록 연결하는 부분과 관련된 설명으로 **가장 적절한** 것은?



1. QPushButton 객체는 setText() 메서드를 사용하여 클릭 이벤트를 감지하고 처리한다.
2. self.button.clicked 는 버튼의 외형을 변경하는 메서드이며, connect() 는 이를 화면에 그리는 역할을 한다.
3. connect() 메서드는 시그널(signal)을 슬롯(slot)에 연결하여, 특정 이벤트(예: 버튼 클릭)가 발생했을 때 지정된 함수를 호출하도록 한다.





4. 버튼의 작동은 QVBoxLayout 에 버튼을 추가하는 순간 자동으로 정의되며, 별도의 연결 과정은 필요하지 않다.
5. on_button_clicked 함수는 QApplication 객체에 직접 연결되어, 애플리케이션 시작과 동시에 한 번만 실행된다.

5. 위에 제시된 PySide6 코드 예시와 관련된 설명으로 **옳지 않은** 것은?

1. 이 코드는 PySide6.QtWidgets 모듈의 클래스들을 활용하여 GUI 애플리케이션을 구축한다.
2. QApplication 객체는 PySide6 애플리케이션의 필수 요소로, 이벤트 루프를 관리하는 역할을 한다.
3. MyWindow 클래스는 QWidget 를 상속받아 커스텀 윈도우를 정의하며, 이 윈도우 안에 QLabel 과 QPushButton 이 포함된다.

4. `self.label.setText("버튼이 클릭되었습니다!")` 코드는 `on_button_clicked` 함수가 실행될 때 라벨의 텍스트를 변경한다.
5. `sys.exit(app.exec())` 대신 `app.exec_()` 를 사용하면 애플리케이션이 종료되지 않고 백그라운드에서 계속 실행된다.  애플리케이션의 이벤트 루프를 시작하고, 애플리케이션이 종료될 때까지 블로킹(blocking) 상태로 유지. 윈도우가 열려 있는 동안 프로그램이 계속 실행되도록 하며, 윈도우가 닫히면 `sys.exit()`를 통해 정상적으로 종료
6. 위에 제시된 Matplotlib 그래프를 추가한 PySide6 코드 예시와 관련된 설명으로 **가장 적절하지 않은** 것은?
 1. `matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg` 클래스는 Matplotlib 그래프를 PySide6 애플리케이션의 위젯으로 통합할 수 있게 해준다.
 2. `self.figure = Figure(...)` 코드는 그래프가 그려질 전체 영역을 담당하는 Matplotlib의 Figure 객체를 생성한다.
 3. `self.axes.clear()` 는 그래프를 업데이트할 때 이전에 그려진 데이터를 지워 새로운 데이터가 겹치지 않도록 하는 역할을 한다.
 4. 버튼을 클릭하여 그래프를 업데이트한 후, 변경된 그래프를 화면에 표시하기 위해서는 `self.canvas.update()` 메서드를 호출해야 한다.
 5. `random` 모듈은 매번 버튼 클릭 시 그래프에 표시될 임의의 데이터를 생성하는 데 사용된다.
7. 파이썬 프로그램을 단일 실행 파일로 만들거나 독립적인 애플리케이션으로 배포하는 것과 관련된 설명으로 **가장 적절하지 않은** 것은?
 1. **PyInstaller, cx_Freeze, py2exe**와 같은 도구는 파이썬 스크립트와 필요한 파이썬 인터프리터, 라이브러리 등을 함께 묶어 독립적인 실행 파일(예: `.exe` 파일)로 만들어준다.
 2. 이러한 "실행 파일 변환" 도구들은 주로 사용자가 파이썬 환경이 설치되어 있지 않은 시스템에서도 프로그램을 실행할 수 있도록 배포 편의성을 높이는 데 사용된다.
 3. `--onefile` 옵션을 사용하여 단일 실행 파일을 생성하는 경우, 모든 종속성이 하나의 파일로 압축되므로 실행 시 압축 해제 과정이 필요 없어 프로그램 시작 속도가 항상 더 빠르다.  실행 시 내부적으로 임시 디렉토리에 압축을 해제하는 과정이 포함될 수 있어 오히려 일반적인 Python 스크립트 실행이나 멀티 파일 빌드보다 시작 속도가 느려질 수 있음.
 4. PyInstaller와 같은 도구로 생성된 실행 파일은 일반적으로 해당 실행 파일을 빌드한 운영체제(예: Windows에서 빌드하면 Windows용 `.exe`)에서만 직접 실행 가능하며, 다른 운영체제에서는 다시 빌드해야 한다.
 5. 단일 실행 파일로 만들지 않고 독립적인 파이썬 애플리케이션을 배포하는 방법으로는 가상 환경(Virtual Environment)을 함께 제공하여 필요한 라이브러리 의존성을 관리하는 방식도 있다.
8. 파이썬 생태계에서 백엔드 개발, 특히 API 등을 제공하는 서버를 구축하는 데 널리 사용되는 웹 프레임워크들에 대한 설명으로 **가장 적절하지 않은** 것은?

1. Django: "배터리 포함(Batteries-included)" 철학을 가진 풀스택(Full-stack) 프레임워크로, ORM, 관리자 페이지 등 다양한 내장 기능을 제공하여 복잡하고 데이터베이스 중심적인 웹 애플리케이션 개발에 강점을 보인다.
 2. Flask: 가볍고 유연한 마이크로 프레임워크(Micro-framework)로, 개발자가 데이터베이스, ORM, 인증 등 핵심 기능을 직접 선택하고 조합할 수 있어 소규모 API나 커스터마이징이 많이 필요한 프로젝트에 적합하다.
 3. FastAPI: 현대적인 고성능 웹 프레임워크로, ASGI(Asynchronous Server Gateway Interface)를 기반으로 비동기(Asynchronous) 처리를 지원하며, 파이썬 타입 힌트를 활용하여 자동 API 문서화(Swagger UI, ReDoc)를 제공하는 것이 큰 특징이다.
 4. Pyramid: 매우 경량의 단일 파일 프레임워크로, 외부 의존성이 거의 없어 빠르고 간단한 API 프로토타이핑에 주로 사용되며, Flask보다 훨씬 더 제한적인 기능을 제공한다.  유연한 확정성이 뛰어난 프레임워크, 극단적인 경량 프레임워크는 아니다. 이는 Bottle
 5. Django와 FastAPI는 모두 대규모 애플리케이션 개발에 사용될 수 있지만, Django는 전통적인 동기(Synchronous) 웹 서비스에 강점을 보이며 FastAPI는 비동기 처리와 높은 동시성을 요구하는 API 서비스에 더 유리하다.
9. FastAPI를 사용하여 인공지능 모델을 서비스하는 간단한 API 프레임워크를 개발할 때 얻을 수 있는 주요 이점 또는 특징으로 가장 적절하지 않은 것은?
1. Pydantic을 활용한 자동 데이터 유효성 검사 및 직렬화: 모델의 입력 및 출력 데이터 스키마를 파이썬 타입 힌트로 정의하고, 이를 기반으로 요청 유효성 검사와 응답 직렬화를 자동으로 처리하여 개발 시간을 단축한다.
 2. ASGI 기반의 비동기 처리 지원: AI 모델 추론과 같이 시간이 소요되는 작업을 비동기적으로 처리하면서도, 다른 클라이언트 요청을 동시에 처리할 수 있어 높은 동시성과 처리량을 제공한다.
 3. 내장된 자동 API 문서화: OpenAPI 표준을 기반으로 Swagger UI와 ReDoc을 자동으로 생성하여, API 명세서 작성 시간을 절약하고 모델을 사용하는 개발자들의 편의성을 높인다.
 4. TensorFlow, PyTorch와 같은 머신러닝 프레임워크의 모델 학습 루틴을 직접 통합하고 관리하는 풍부한 내장 유틸리티를 제공한다. 
 5. 의존성 주입(Dependency Injection) 시스템: 모델 로딩, 데이터베이스 연결, 인증 토큰 검증 등 API 엔드포인트에 필요한 복잡한 종속성을 효율적으로 관리하고 재사용성을 높인다.
- 다음은 간단한 FastAPI 코드 보기이다.

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
```

```

from typing import List, Dict

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None

db: List[Item] = []

@app.get("/")
async def read_root():
    """ 애플리케이션의 루트 경로입니다. """
    return {"message": "Hello, FastAPI! Welcome to the AI Model API."}

@app.get("/items/", response_model=List[Item])
async def read_items():
    """ 모든 아이템 목록을 조회합니다. """
    return db

@app.get("/items/{item_id}", response_model=Item)
async def read_item(item_id: int):
    """ ID를 통해 특정 아이템을 조회합니다. """
    if item_id < 0 or item_id >= len(db):
        raise HTTPException(status_code=404, detail="Item not found")
    return db[item_id]


@app.post("/items/", response_model=Item, status_code=201)
async def create_item(item: Item):
    """ 새로운 아이템을 생성합니다. """
    db.append(item)
    return item

```

10. 위에 제시된 FastAPI 코드 예시와 그 기능에 대한 설명으로 가장 적절하지 않은 것은?

1. `@app.get("/")` 데코레이터는 클라이언트가 루트 경로 (/)로 HTTP GET 요청을 보냈을 때 `read_root` 함수가 실행되도록 연결한다.
2. `Item(BaseModel)` 클래스는 클라이언트가 `/items/` 경로로 POST 요청을 보낼 때 전송하는 데이터의 구조와 타입을 정의하며, 이 정의를 기반으로 자동적인 데이터 유효성 검사가 이루어진다.
3. `/items/{item_id}` 경로에서 `item_id: int` 와 같이 타입 힌트를 사용함으로써, FastAPI는 해당 경로 매개변수가 정수형이어야 함을 자동으로 검증하고, 유효하지 않을 경우 오류를 반환한다.
4. `read_item` 함수에서 `raise HTTPException(status_code=404, detail="Item not found")` 는 특정 아이템을 찾지 못했을 때 404 Not

Found HTTP 상태 코드와 함께 오류 메시지를 클라이언트에 전달하는 역할을 한다.

5. `create_item` 함수는 클라이언트로부터 받은 `Item` 객체를 `db` 리스트에 추가한 후, 응답 모델(`response_model=Item`)이 지정되어 있으므로 클라이언트에게는 추가된 아이템의 `name` 필드만 반환한다. 

`response_model=Item`은 API 응답이 `Item` 모델의 스키마를 따를 것임을 선언하는 것. `return item`을 수행하면 클라이언트에게 추가된 아이템 객체 `Item` 모델에 정의된 모든 필드가 JSON 형태로 반환