FastAPI를 이용한 파이썬 웹 애플리케이션

웹 애플리케이션 프로젝트 만들기

FastAPI를 이용한 파이썬 웹 애플리케이션

- 파이썬 개발환경 구성
- FastAPI를 이용한 웹 애플리케이션

1절. FastAPI를 이용한 파이썬 웹 애플리케이션

1장. 웹 애플리케이션 프로젝트 만들기

1.1. 개발환경

Python 인터프리터

https://www.python.org

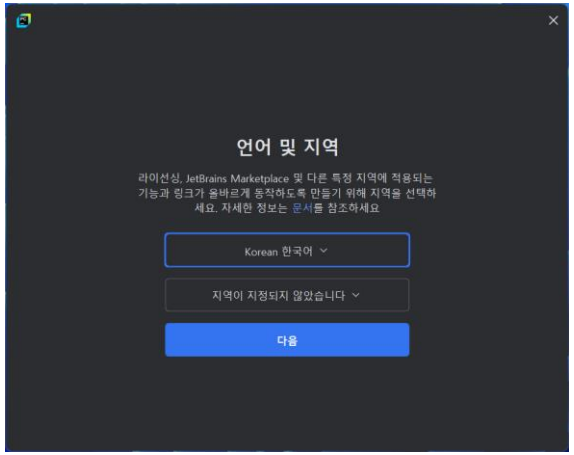
설치 시 "Add Python to PATH" 옵션을 선택

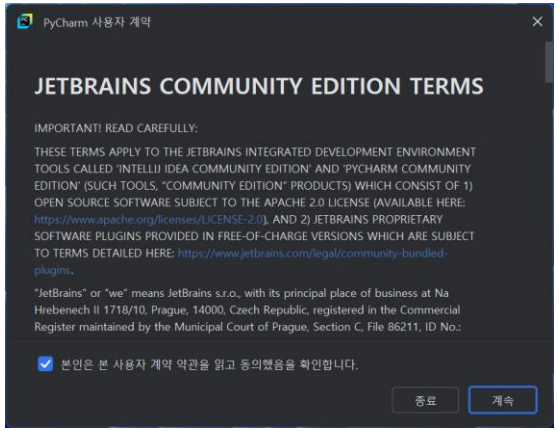
- 파이썬은 3.12 버전을 내려받으세요.
- https://www.python.org/downloads/release/python-3128/

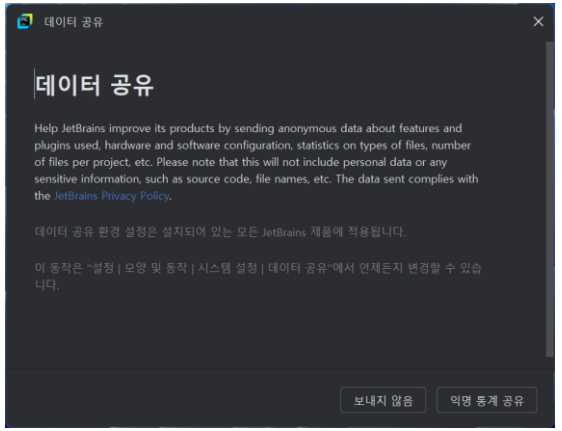
PyCharm

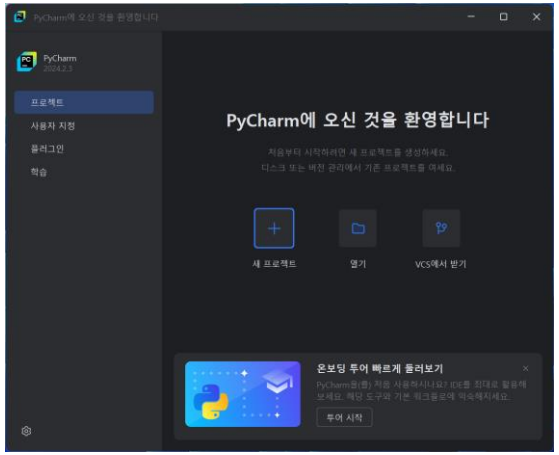
https://www.jetbrains.com/ -> Developer Tools -> PyCharm -> PyCharm Community Edition

https://www.jetbrains.com/pycharm/ -> PyCharm Community Edition









1.2. FastAPI 설치


```
pip install fastapi uvicorn
```

ASGI

- ASGI(Asynchronous Server Gateway Interface)
- 파이썬에서 비동기 웹 서버와 웹 애플리케이션 간의 인터페이스 표준.
- ASGI는 기존 WSGI(Web Server Gateway Interface)의 비동기 버전임.
- 파이썬에서 비동기 처리를 지원하는 웹 애플리케이션을 구축하기 위해 설계되었음.

ASGI의 주요 특징

- **비동기 지원**: ASGI는 비동기 코드 실행을 지원하여 높은 성능과 동시성을 제공합니다. 이는 특히 웹소켓이나 서버 푸시와 같은 비동기 통신이 필요한 애플리케이션에 유용합니다.
- **범용성**: ASGI는 HTTP뿐만 아니라 WebSocket, gRPC와 같은 다른 프로토콜도 지원합니다.
- **유연성**: ASGI 애플리케이션은 다양한 서버 및 프레임워크와 호환되며, 모듈식으로 구성할 수 있습니다.

FastAPI와 ASGI

- FastAPI는 ASGI 표준을 따르는 웹 프레임워크
- FastAPI 애플리케이션은 비동기 처리를 기본으로 하며, Uvicorn과 같은 ASGI 서버를 사용하여 높은 성능을 제공함

1.3. FastAPI 애플리케이션 생성

main.py

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def index():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
async def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}

```

- `/`: 루트 경로에 대한 GET 요청을 처리합니다.
- `/items/{item_id}`: 동적 경로 매개변수(Path Parameters) item_id를 사용하여 특정 아이템을 조회합니다. 경로 매개변수는 중괄호 `{}` 안에 변수명을 적어 정의합니다.

FastAPI는 경로 매개변수와 쿼리 매개변수를 함께 사용할 수 있습니다. 경로 매개변수는 URL 경로의 일부로 사용되고, 쿼리 매개변수는 URL의 쿼리 문자열로 전달됩니다.

- item_id: 경로 매개변수입니다.
- q: 쿼리 매개변수(기본값은 None)입니다.

1.4. 데이터 모델링

main.py

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


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


@app.post("/items/")
async def create_item(item: Item):
    return item

```

- BaseModel은 pydantic 라이브러리의 핵심 클래스입니다.
- pydantic은 데이터 유효성 검사와 설정 관리에 사용되는 파이썬 라이브러리로, 데이터 모델링을 쉽고 강력하게 할 수 있도록 도와줍니다.
- FastAPI는 pydantic을 사용하여 데이터 유효성 검사를 수행합니다.

- 7라인에 정의된 클래스를 15라인의 앱 함수 매개변수로 설정하면 FastAPI가 요청 본문을 자동으로 Item 모델로 변환하고, 유효성 검사를 수행합니다. 잘못된 데이터가 입력되면 자동으로 `422 Unprocessable Entity` 응답을 반환합니다.

1.5. FastAPI 문서화

- FastAPI는 자동으로 API 문서를 생성합니다.
- 애플리케이션을 실행한 후 브라우저에서 다음 URL을 방문하여 API 문서를 확인할 수 있습니다.
  - Swagger UI: http://127.0.0.1:8000/docs
  - ReDoc: http://127.0.0.1:8000/redoc

Swagger UI와 Redoc는 API 문서화를 위한 도구입니다. 이들은 RESTful API의 문서와 사용 방법을 시각적으로 표현하여 개발자들이 쉽게 이해하고 테스트할 수 있도록 도와줍니다.

main.py

```python
from fastapi import FastAPI

app = FastAPI(
    title="My API",
    description="This is a sample API",
    version="1.0.0",
    docs_url=None,        # Swagger UI 비활성화
    redoc_url=None        # ReDoc 비활성화
)
```

FastAPI는 기본적으로 Swagger UI와 ReDoc을 활성화합니다. 하지만 원하는 경우 왼쪽의 코드처럼 이를 비활성화하거나 커스터마이징할 수 있습니다.

1.6. FastAPI 미들웨어

FastAPI 미들웨어

- 요청(request)과 응답(response) 사이에서 특정 작업을 수행하는 데 사용되는 함수 또는 클래스
- 모든 요청에 대해 실행되며, 요청을 처리하기 전에 또는 응답을 반환하기 전에 특정 작업을 수행할 수 있음
- 예를 들어, 로깅, 인증, CORS 처리, 압축 등이 미들웨어를 통해 구현될 수 있음

```
5  app = FastAPI()
6
7
8  class LoggingMiddleware(BaseHTTPMiddleware):
9      async def dispatch(self, request, call_next):
10         logging.info(f"Req: {request.method} {request.url}")
11         response = await call_next(request)
12         logging.info(f"Status code: {response.status_code}")
13         return response
14
15 app.add_middleware(LoggingMiddleware)
```

1.7. FastAPI 종합 예제

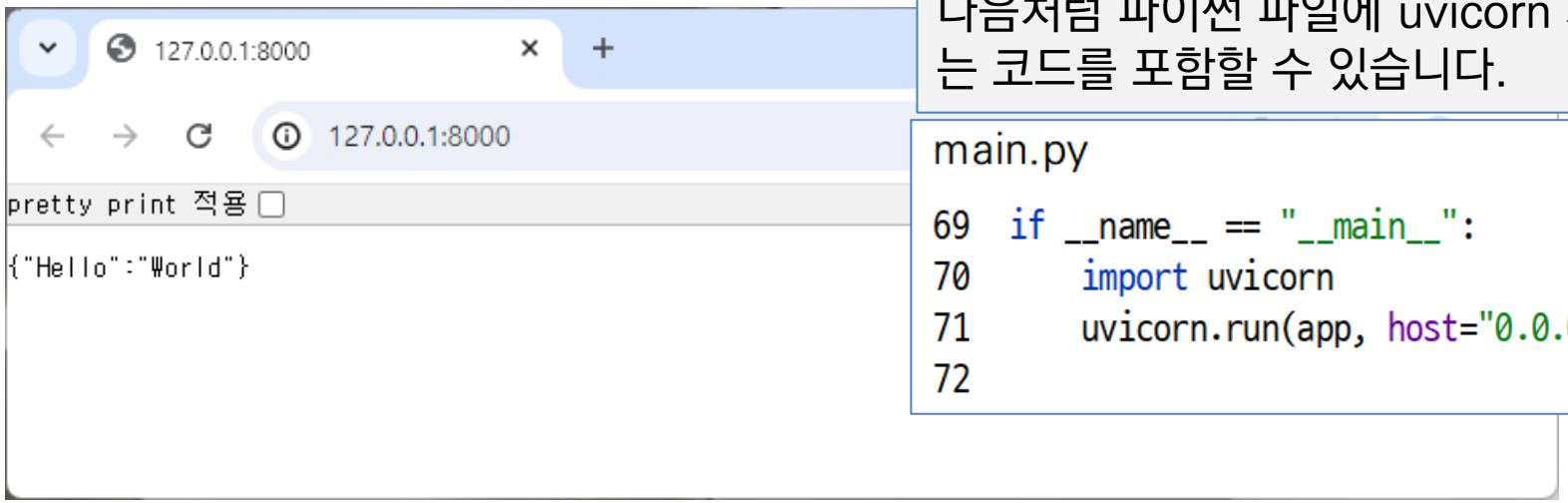
main.py

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

app = FastAPI()

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

items = {}

@app.get("/")
async def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
async def read_item(item_id: int):
    if item_id not in items:
        raise HTTPException(status_code=404, detail="Item not found")
    return items[item_id]

@app.post("/items/")
async def create_item(item: Item):
    item_id = len(items) + 1
    items[item_id] = item
    return {"item_id": item_id, **item.dict()}

```

더 많은 기능과 세부 설정은 FastAPI 공식 문서(https://fastapi.tiangolo.com/)를 참고하세요.

1.8. 애플리케이션 실행


```
uvicorn main:app --reload
```



다음처럼 파이썬 파일에 uvicorn 서버를 실행하는 코드를 포함할 수 있습니다.

main.py

```
69  if __name__ == "__main__":
70      import uvicorn
71      uvicorn.run(app, host="0.0.0.0", port=8000)
72
```

서버의 실행되는 포트를 바꾸고 싶다면 아래처럼 --port 옵션을 추가하세요. 다음은 서버의 포트를 8001번으로 지정하는 예입니다.

```
uvicorn main:app --host 0.0.0.0 --port 8001
```