파일 요청

`File`을 사용하여 클라이언트가 업로드할 파일들을 정의할 수 있습니다.

!!! info "정보" 업로드된 파일을 전달받기 위해 먼저 `python-multipart`를 설치해야합니다.

```
예시) `pip install python-multipart`.

업로드된 파일들은 "폼 데이터"의 형태로 전송되기 때문에 이 작업이 필요합니다.
```

`File` 임포트

`fastapi`에서 `File`과 `UploadFile`을 임포트 합니다:

```
{!../../../docs_src/request_files/tutorial001.py!}
```

`File` 매개변수 정의

`Body` 및 `Form`과 동일한 방식으로 파일의 매개변수를 생성합니다:

```
{!../../../docs_src/request_files/tutorial001.py!}
```

!!! info "정보" `File`은 `Form`으로부터 직접 상속된 클래스입니다.

```
하지만 `fastapi`로부터 `Query`, `Path`, `File` 등을 임포트 할 때, 이것들은 특별한 클래스들을 반환하는 함수라는 것을 기억하기 바랍니다.
```

!!! tip "팁" File의 본문을 선언할 때, 매개변수가 쿼리 매개변수 또는 본문(JSON) 매개변수로 해석되는 것을 방지하기 위해 `File`을 사용해야합니다.

파일들은 "폼 데이터"의 형태로 업로드 됩니다.

*경로 작동 함수*의 매개변수를 `bytes`로 선언하는 경우 **FastAPI**는 파일을 읽고 `bytes` 형태의 내용을 전달합니다.

이것은 전체 내용이 메모리에 저장된다는 것을 의미한다는 걸 염두하기 바랍니다. 이는 작은 크기의 파일들에 적합합니다.

어떤 경우에는 `UploadFile`을 사용하는 것이 더 유리합니다.

`File` 매개변수와 `UploadFile`

`File` 매개변수를 `UploadFile` 타입으로 정의합니다:

```
{!../../../docs_src/request_files/tutorial001.py!}
```

`UploadFile`을 사용하는 것은 `bytes`과 비교해 다음과 같은 장점이 있습니다:

- "스풀 파일"을 사용합니다.
    - 최대 크기 제한까지만 메모리에 저장되며, 이를 초과하는 경우 디스크에 저장됩니다.

- 따라서 이미지, 동영상, 큰 이진코드와 같은 대용량 파일들을 많은 메모리를 소모하지 않고 처리하기에 적합합니다.
- 업로드 된 파일의 메타데이터를 얻을 수 있습니다.
- file-like `async` 인터페이스를 갖고 있습니다.
- file-like object를 필요로하는 다른 라이브러리에 직접적으로 전달할 수 있는 파이썬 `SpooledTemporaryFile` 객체를 반환합니다.

`UploadFile`

`UploadFile` 은 다음과 같은 어트리뷰트가 있습니다:

- `filename` : 문자열(`str`)로 된 업로드된 파일의 파일명입니다 (예: `myimage.jpg`).
- `content_type` : 문자열(`str`)로 된 파일 형식(MIME type / media type)입니다 (예: `image/jpeg`).
- `file` : `SpooledTemporaryFile` (파일류 객체)입니다. 이것은 "파일류" 객체를 필요로하는 다른 라이브러리에 직접적으로 전달할 수 있는 실질적인 파이썬 파일입니다.

`UploadFile` 에는 다음의 `async` 메소드들이 있습니다. 이들은 내부적인 `SpooledTemporaryFile` 을 사용하여 해당하는 파일 메소드를 호출합니다.

- `write(data)` : `data` (`str` 또는 `bytes`)를 파일에 작성합니다.
- `read(size)` : 파일의 바이트 및 글자의 `size` (`int`)를 읽습니다.
- `seek(offset)` : 파일 내 `offset` (`int`) 위치의 바이트로 이동합니다.
  - 예) `await myfile.seek(0)` 를 사용하면 파일의 시작부분으로 이동합니다.
  - `await myfile.read()` 를 사용한 후 내용을 다시 읽을 때 유용합니다.
- `close()` : 파일을 닫습니다.

상기 모든 메소드들이 `async` 메소드이기 때문에 "await"을 사용하여야 합니다.

예를들어, `async` *경로 작동 함수*의 내부에서 다음과 같은 방식으로 내용을 가져올 수 있습니다:

```
contents = await myfile.read()
```

만약 일반적인 `def` *경로 작동 함수*의 내부라면, 다음과 같이 `UploadFile.file` 에 직접 접근할 수 있습니다:

```
contents = myfile.file.read()
```

!!! note " `async` 기술적 세부사항" `async` 메소드들을 사용할 때 **FastAPI**는 스레드풀에서 파일 메소드들을 실행하고 그들을 기다립니다.

!!! note "Starlette 기술적 세부사항" **FastAPI**의 `UploadFile` 은 **Starlette**의 `UploadFile` 을 직접적으로 상속받지만, **Pydantic** 및 FastAPI의 다른 부분들과의 호환성을 위해 필요한 부분들이 추가되었습니다.

"폼 데이터"란

HTML의 폼들(`<form></form>`)이 서버에 데이터를 전송하는 방식은 대개 데이터에 JSON과는 다른 "특별한" 인코딩을 사용합니다.

FastAPI는 JSON 대신 올바른 위치에서 데이터를 읽을 수 있도록 합니다.

!!! note "기술적 세부사항" 폼의 데이터는 파일이 포함되지 않은 경우 일반적으로 "미디어 유형" `application/x-www-form-urlencoded` 을 사용해 인코딩 됩니다.

하지만 파일이 포함된 경우, `multipart/form-data`로 인코딩됩니다. `File`을 사용하였다면, **FastAPI**는 본문의 적합한 부분에서 파일을 가져와야 한다는 것을 인지합니다.

인코딩과 폼 필드에 대해 더 알고싶다면, <code>POST</code>에 관한<abbr title="Mozilla Developer Network">MDN</abbr>웹 문서 를 참고하기 바랍니다,.

!!! warning "주의" 다수의 `File` 과 `Form` 매개변수를 한 *경로 작동*에 선언하는 것이 가능하지만, 요청의 본문이 `application/json` 가 아닌 `multipart/form-data` 로 인코딩 되기 때문에 JSON으로 받아야하는 `Body` 필드를 함께 선언할 수는 없습니다.

이는 **FastAPI**의 한계가 아니라, HTTP 프로토콜에 의한 것입니다.

다중 파일 업로드

여러 파일을 동시에 업로드 할 수 있습니다.

그들은 "폼 데이터"를 사용하여 전송된 동일한 "폼 필드"에 연결됩니다.

이 기능을 사용하기 위해, `bytes` 의 `List` 또는 `UploadFile` 를 선언하기 바랍니다:

```
{!../../../docs_src/request_files/tutorial002.py!}
```

선언한대로, `bytes` 의 `list` 또는 `UploadFile` 들을 전송받을 것입니다.

!!! note "참고" 2019년 4월 14일부터 Swagger UI가 하나의 폼 필드로 다수의 파일을 업로드하는 것을 지원하지 않습니다. 더 많은 정보를 원하면, #4276과 #3641을 참고하세요.

그럼에도, **FastAPI**는 표준 OpenAPI를 사용해 이미 호환이 가능합니다.

따라서 Swagger UI 또는 기타 그 외의 OpenAPI를 지원하는 툴이 다중 파일 업로드를 지원하는 경우, 이들은 **FastAPI**와 호환됩니다.

!!! note "기술적 세부사항" `from starlette.responses import HTMLResponse` 역시 사용할 수 있습니다.

FastAPI는 개발자의 편의를 위해 `fastapi.responses` 와 동일한 `starlette.responses` 도 제공합니다. 하지만 대부분의 응답들은 Starlette로부터 직접 제공됩니다.

요약

폼 데이터로써 입력 매개변수로 업로드할 파일을 선언할 경우 `File` 을 사용하기 바랍니다.