01 FastAPI Intro

Python Decorators and FastAPI

Understanding Decorators in Python

Decorators are a powerful feature in Python that allow you to modify the behavior of functions or methods without changing their source code. They use the `@` syntax and are essentially functions that take another function as an argument and extend its functionality.

Basic Decorator Structure

```python
def my_decorator(func):
    def wrapper(*args, **kwargs):
        # Code to execute before the function
        result = func(*args, **kwargs)  # Call the original function
        # Code to execute after the function
        return result
    return wrapper

@my_decorator
def my_function():
    # Function implementation
```

Example: Timing Decorator

```python
import time

def timer(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"The total time it took to run is {end - start:.2f} s")
        return result
    return wrapper

@timer
def slow_func(something):
    print("started ... ")
    time.sleep(2.5)
    print("completed ... ")
    print(something)
```

Important Points About Decorators

- The inner function (commonly named `wrapper`, but can be any name) acts as a replacement for the original function
- Always return the inner function from your decorator

- Pass `*args` and `**kwargs` to both capture and forward any arguments to the original function
- Return the result of the original function if you want to preserve its return value

Creating a Simple API with FastAPI

FastAPI is a modern Python web framework for building APIs with minimal code. It leverages Python type hints for validation and documentation.

Basic Setup

```
# FastAPI is a class we import from the fastapi module
from fastapi import FastAPI

# Create an app instance
# This app variable is the main point of interaction to create all your API
endpoints
app = FastAPI()
```

Creating Endpoints

```
# @ is the path operation decorator
# '/' refers to the root path (last part of URL)
# Path is also called an endpoint or route
# get is an HTTP method (operation)
@app.get('/')
def index():
    return 'Hello world'

# Handling another path
@app.get('/property')
def property():
    return 'This is the property page'

# Returning JSON instead of a simple string
@app.get('/movies')
def movies():
    return {'movie list': ['movie 1', 'movie 2', 'movie 3']}
```

Running the Application

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

Key Concepts

- **Path Operation Decorator**: The `@app.get('/')` syntax is a decorator that tells FastAPI which function should handle requests to a specific path with a specific HTTP method
- **Path/Endpoint/Route**: The URL path that the client will request (e.g., `/movies`)
- **Operation**: The HTTP method (GET, POST, PUT, DELETE, etc.)
- **Path Operation Function**: The function that will be called by FastAPI when it receives a request to a specific path using a specific HTTP method

FastAPI automatically converts return values (dictionaries, lists, strings) to JSON responses when appropriate.

What is Uvicorn?

Uvicorn is an **ASGI server**. Think of it like a waiter in a restaurant — it takes requests from the outside world (like someone ordering food), passes them to your FastAPI app (your kitchen), and then returns the response (the food).

FastAPI apps **don't run by themselves**. They need a server like Uvicorn to handle incoming HTTP requests and send responses back.

ASGI vs WSGI (quick note)

- **WSGI** is the older standard (used by Flask, Django).
- **ASGI** is newer and supports **async** operations — great for real-time stuff, WebSockets, etc.
- FastAPI is built for **ASGI**, and Uvicorn is an **ASGI-compatible server**.

How do you use Uvicorn?

Let's say you've made this `main.py` file:

```
from fastapi import FastAPI

app = FastAPI()

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

To run this using Uvicorn:

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

- `main` is the Python file name (without `.py`)
- `app` is the FastAPI instance (`app = FastAPI()`)
- `--reload` auto-restarts the server when code changes — great for development

---

Why not just `python main.py`?

Because FastAPI doesn't have a built-in HTTP server. It's like writing the kitchen code but having no one to take orders. Uvicorn steps in to serve your app to the world.

Deep dive? -> https://medium.com/@mamtag962000/uvicorn-7f28a3016cc4