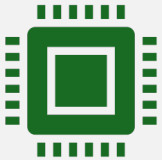
Advanced Python

Surendra Panpaliya

Week1 (Mon, Tue, Thurs)

Day 1: Python Recap + Environment & Tooling



Day 2: Functional Programming & Object-Oriented Design

Day 3: Advanced Python Concepts

Week2 (Mon, Tue, Wed, Thurs)

Day 4: Concurrency and Async Programming

Day 5: Web Services with FastAPI

Day 6: Azure Functions & Cloud Deployment

Day 7: Testing, Linting & Final Project

Day 5: Web Services with FastAPI

- FastAPI project structure vs ASP.NET Core WebAPI
- Path/query parameters, request validation
- Dependency Injection (DI)
- Serving with Uvicorn + Gunicorn

Day 5: Web Services with FastAPI

Hands-On Lab:

Build CRUD APIs using FastAPI

ASP.NET Route attributes vs FastAPI decorators

FastAPI

What is FastAPI?

FASTAPI IS A MODERN,

FAST (HIGH-PERFORMANCE),

WEB FRAMEWORK FOR BUILDING APIS

WITH PYTHON 3.7+ BASED

ON STANDARD PYTHON TYPE HINTS.

What is FastAPI?

FastAPI is a web framework

Allows you to build RESTful APIs

easily and quickly.

https://fastapi.tiangolo.com/

What is FastAPI?

Designed for building web APIs,

especially microservices,

built on top of:

Starlette (for web handling, routing, etc.)

Pydantic (for data validation and settings management)

Starlette

Lightweight ASGI

(Asynchronous Server Gateway Interface)

framework and toolkit

for building high-performance

async web applications and

APIs in Python.

Pydantic

PYDANTIC IS A DATA VALIDATION AND

DATA PARSING LIBRARY BASED ON

PYTHON TYPE HINTS.

Python Type Hints

- A way to explicitly declare
- the expected data types of variables,
- function parameters
- return values.
- Do not enforce types at runtime.

Function with Type Hints

```
def add(x: int, y: int) -> int:
return x + y
```

- x: int → x should be an integer
- y: int → y should be an integer
- -> int → function returns an integer

⚙️ Key Features of FastAPI

Surendra Panpaliya

1. Blazing Fast Performance

Built on Starlette and Pydantic

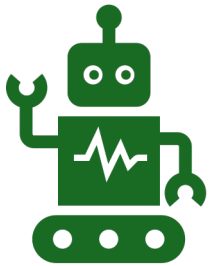
FastAPI delivers performance comparable

to Node.js and Go,

making it one of the fastest Python frameworks available.

2. ⚡ Fast to Code

Boosts development speed by 200–300%,



thanks to automatic validation,

smart tooling, and minimal boilerplate code.

3. Fewer Bugs

Reduces human (developer) errors

by up to 40%,

with built-in type validation,

static checking, and

structured data models.

4. Intuitive and Developer-Friendly

Excellent editor support with features

like auto-completion,

inline error detection, and

type-aware suggestions

for faster and smarter development.

5. Easy to Learn & Use

CLEAN SYNTAX, WELL-STRUCTURED DOCS,

AND AUTOMATIC API DOCS (SWAGGER UI, REDOC)

MAKE IT IDEAL FOR BEGINNERS AND

PROFESSIONALS ALIKE.

6. Short & Clean Codebase

ELIMINATES CODE DUPLICATION

BY USING DECLARATIVE PROGRAMMING

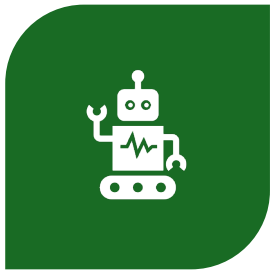
FOR REQUEST PARSING,

RESPONSE MODELING, AND

DEPENDENCY INJECTION.

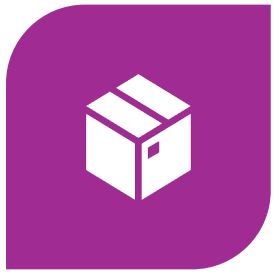
7. Robust and Production-Ready

SHIPS WITH INTERACTIVE DOCUMENTATION,



AUTOMATIC VALIDATION, ERROR HANDLING, AND

SEAMLESS SUPPORT FOR OAUTH2/JWT



– ALL OUT OF THE BOX.

8. Standards-Compliant

FULLY BASED ON OPEN STANDARDS:

OPENAPI (FORMERLY SWAGGER) FOR API DEFINITION

JSON SCHEMA FOR DATA VALIDATION AND INTEROPERABILITY

OpenAPI

A standard specification for describing REST APIs.

Defines what endpoints exist,

what data they accept, and what they return.

Swagger UI

A web-based interface

Renders OpenAPI docs in an interactive way.

Allows you to try out API endpoints

directly from the browser.

FastAPI project structure vs ASP.NET Core WebAPI

Feature	FastAPI (Python)	ASP.NET Core WebAPI (C#)
Language	Python	C#
Framework Type	Minimalistic, async-first web API	Full-featured enterprise web framework
Performance	Very fast (Starlette + Uvicorn)	Very fast (compiled + async)
Use Case	APIs, microservices, ML/AI backend	Enterprise APIs, large-scale systems

FastAPI project structure vs ASP.NET Core WebAPI

Feature	FastAPI (Python)	ASP.NET Core WebAPI (C#)
Entry Point	main.py	Program.cs, Startup.cs
Routing	APIRouter with decorators	[ApiController] with [HttpGet]
Models	Pydantic	C# POCO Classes
Dependency Injection	Via Depends	Built-in DI container

FastAPI project structure vs ASP.NET Core WebAPI

Feature	FastAPI (Python)	ASP.NET Core WebAPI (C#)
Async Support	Native async with async def	async Task<IActionResult>
Performance	Very high (Uvicorn)	Very high (Kestrel)
Hot Reload	uvicorn --reload	dotnet watch run
Use Case	APIs, ML microservices	Enterprise APIs, .NET ecosystems

When to Use Which?

Use Case	Use FastAPI	Use ASP.NET Core WebAPI
Lightweight microservices	✅ Yes	✅ Yes
Deep integration with ML/AI models	✅ Strong Python support	❌ Limited
Enterprise-scale architecture	⚠️ Can scale, but lighter	✅ Ideal for large systems
Corporate/Windows ecosystem	❌	✅ Full support
Async-first APIs	✅ Fully supported	✅ Supported since .NET Core

Path/query parameters, request validation

Parameter Type	Used For	Example URL
Path Parameter	Identifies a resource	/users/123
Query Parameter	Filters, sorts, modifies result	/users?age=25&active=true

Path/query parameters, request validation

Feature	FastAPI (Python)	ASP.NET Core WebAPI (C#)
Path Parameters	@app.get("/items/{id}")	[HttpGet("items/{id}")]
Query Parameters	Function defaults / Query() from FastAPI	[FromQuery] annotation
Request Body Validation	Pydantic models	C# Models + Data Annotations
Auto error responses	✅ Built-in	✅ With ModelState.IsValid check
OpenAPI Schema	✅ Auto-generated	✅ Via Swashbuckle/Swagger
Default Values	def f(limit: int = 10)	[FromQuery] int limit = 10

Use Cases

Use Case	Path	Query	Body
Get user by ID	✅	❌	❌
Search/filter user list	❌	✅	❌
Create new user	❌	❌	✅
Paginated resource	✅	✅	❌

Dependency Injection (DI)

Surendra Panpaliya

What is Dependency Injection (DI)?

Design pattern used to achieve

Inversion of Control (IoC)

by injecting dependent objects (services)

into a class instead of

creating them inside the class.

Why use DI?

Decouples business logic from instantiation logic

Increases testability (easy to mock)

Promotes maintainability and scalability

Why Dependency Injection (DI)?

In FastAPI, Depends() is used for

dependency injection

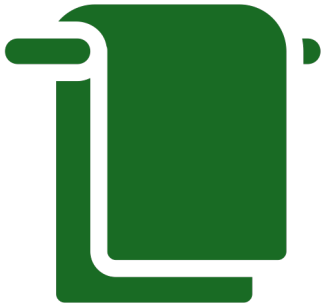
Centralizing shared logic

Ex. database calls

authentication

Why Dependency Injection (DI)?

Reducing code duplication



(DRY principle)

Why Dependency Injection (DI)?

Making code more

modular,

reusable,

testable, and

maintainable

Key Terms

Term	Meaning
Dependency	A class or object your code depends on
Injection	Supplying that dependency externally
IoC Container	Framework that manages object lifecycle and dependencies

💡 FastAPI DI Summary

Feature	FastAPI
DI Mechanism	Depends()
Scope Control	Via yield/return in dependency
Advanced Usage	Sub-dependencies, caching, security
Testing	Easily mock with TestClient

Testing With DI

Feature	FastAPI	ASP.NET Core
Override DI	app.dependency_overrides[]	Use mocks with TestServer
Unit Testing	Inject fake service	Mock interfaces with Moq

FastAPI vs ASP.NET Core DI

Feature	FastAPI (Python)	ASP.NET Core WebAPI (C#)
DI Style	Function-based (Depends)	Constructor-based
Container Setup	Implicit (via decorators)	Explicit (builder.Services)
Lifetime Control	Via generator/yield	Scoped, Transient, Singleton
Common Use Cases	DB session, auth, services	Services, repositories, logging
Testing Support	Easy via overrides	Strong via interfaces + Moq

Real-World Use Cases

Use Case	Dependency Example
Logging or Metrics	Inject Logger or Tracer
Database access	Inject DB session or repository
Configs/API keys	Inject config object
Authenticated user info	Inject current user from token
Background tasks	Inject queue or thread-pool manager

Serving with Uvicorn + Gunicorn

Surendra Panpaliya

1. What are Uvicorn and Gunicorn?

Tool	Role	Description
Uvicorn	ASGI server	Runs asynchronous Python web apps (FastAPI, Starlette)
Gunicorn	WSGI/Process manager	Manages multiple Uvicorn workers (processes) for high-concurrency

1. What are Uvicorn and Gunicorn?

Uvicorn alone is good for development

Uvicorn + Gunicorn is recommended for **production**

2. Why Use Gunicorn with Uvicorn?

Concern	Uvicorn Alone	Uvicorn + Gunicorn
Concurrent Users	Limited to single process	Multi-process handling via Gunicorn
CPU Core Utilization	Only 1 core	Gunicorn spawns multiple workers per core
Graceful Reloading	Limited	Robust reloading, logging, process management
Recommended For	Development	Production (e.g., with Docker, Nginx)

3. Install Requirements

pip install fastapi uvicorn gunicorn

Run with Uvicorn (DEV ONLY)

uvicorn app.main:app –reload

- --reload is for development
- Not suitable for production due to single-threaded limitation

3. Install Requirements

Run with Gunicorn + Uvicorn Workers (Production)

```
gunicorn app.main:app \
  -k uvicorn.workers.UvicornWorker \
   --workers 4 \
   --bind 0.0.0.0:8000
```

Run with Gunicorn + Uvicorn Workers (Production)

Argument	Purpose
-k UvicornWorker	Tells Gunicorn to use Uvicorn's ASGI worker
--workers 4	Number of worker processes (CPU cores)
--bind 0.0.0.0:8000	Binds the app to a network port

Formula to decide workers

workers = (CPU_CORES * 2) + 1

Uvicorn vs Gunicorn

Feature	Uvicorn	Gunicorn + UvicornWorker
Best for	Development	Production
Multi-core	❌ No (1 process)	✅ Yes (multi-process)
Performance	High	Even higher with concurrency
Command Example	uvicorn main:app --reload	gunicorn main:app -k uvicorn.workers.UvicornWorker

Quick Summary Table

Feature	ASP.NET Core (C#)	FastAPI (Python)
Route Definition	Attributes (e.g., [HttpGet])	Decorators (e.g., @app.get())
Base Route	[Route("api/[controller]")]	app = FastAPI() (no base route)
Path Parameters	In route string with {}	In decorator string with {}
Query Parameters	[FromQuery]	Function parameters with defaults

Quick Summary Table

Feature	ASP.NET Core (C#)	FastAPI (Python)
HTTP Method Mapping	[HttpGet], [HttpPost], etc.	@app.get, @app.post, etc.
Versioning Support	With route prefix or [ApiVersion]	Custom prefix or APIRouter
Automatic Docs	Swagger via Swashbuckle	Swagger UI via built-in OpenAPI

Quick Summary Table

Capability	ASP.NET Core	FastAPI
Route Style	Attribute-based	Decorator-based
Base Routing	Controller-level [Route()]	Use APIRouter
Path/Query Handling	Explicit attributes	Auto-handled with type hints

Quick Summary Table

Capability	ASP.NET Core	FastAPI
Validation	Manual or with FluentValidation	Built-in with Pydantic
Swagger/OpenAPI	Needs setup (Swashbuckle)	Built-in
Dependency Injection	First-class with constructor	Via Depends()

Happy Learning!! Thanks for Your Patience ☺

Surendra Panpaliya

GKTCS Innovations