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 3: Advanced Python Concepts

- Decorators: Logging, validation, chaining
- Context Managers: with statement, __enter__, __exit__
- Generators and yield, pipelines
- Metaclasses: Framework-level magic

Day 3: Advanced Python Concepts

Hands-On Lab:

Logger with decorators and context managers

C# Attributes vs Python Decorators

What is a Decorator?

A **DECORATOR** IS A FUNCTION THAT

TAKES ANOTHER FUNCTION AS INPUT,

ADDS SOME FUNCTIONALITY

RETURNS A NEW FUNCTION

WITHOUT MODIFYING THE ORIGINAL ONE.

What is a Decorator?

Used in Python for:

Cross-cutting concerns

like logging, validation, caching

Reusability and cleaner code

Summary

Decorator Type	Purpose	Example
@log	Logs function calls and return values	Debugging, Monitoring
@validate_positive	Validates inputs before execution	Input sanitization
Chained	Combines multiple behaviors	@log, @validate_positive
With args	Custom behavior	@repeat(n)

Context Managers

with statement

__enter__

__exit__

🧠 What is a Context Manager?

A construct that **sets up a resource,**

does something with it,

and then tears it down — automatically.

🧠 What is a Context Manager?

Used with the with statement

To **manage resources like files,**

Database connections, locks

Ensuring proper cleanup.

Use Cases of Context Managers

Use Case	Example
File handling	open()
Locking	with threading.Lock():
Database connections	with db.connect():
Temporary change	with open_temp_file():
Timing, logging, debugging	Custom context managers

Generators and yield, pipelines

Surendra Panpaliya

What is a Generator?

- A **generator** is a special type of function
- Uses yield instead of return
- **Remembers its state** between calls
- Produces a **sequence of values lazily** (on demand)

🧠 Benefits of Generators

Feature	Benefit
Lazy evaluation	Efficient memory use
Pause & Resume	State is saved automatically
Composable	Can be chained like Unix pipes
Infinite series	Great for streaming or unbounded data

Summary Table

Concept	Python Syntax	Description
Generator function	def func(): yield	Creates a generator
Generator object	gen = func()	Lazily returns next value
Generator loop	for x in gen:	Loops through values
Pipeline chaining	f3(f2(f1(data)))	Builds reusable streams
Generator expr	(x*x for x in range(5))	Inline generator

Metaclasses: Framework-level magic

Surendra Panpaliya

GKTCS Innovations

🧠 What is a Metaclass?

A **METACLASS** IS THE **CLASS OF A CLASS.**

JUST LIKE **A CLASS CREATES OBJECTS,**

A METACLASS CREATES CLASSES.

✅ Why Use Metaclasses?

Allow you to **control class creation,**

just like a class controls object creation

Common Use Cases

ENFORCING CODING STANDARDS

AUTO-REGISTERING CLASSES

(PLUGINS, COMMANDS, SERIALIZERS)

VALIDATING CLASS ATTRIBUTES

Common Use Cases

ADDING METHODS DYNAMICALLY

BUILDING FRAMEWORKS

DJANGO MODELS, SQLALCHEMY TABLES

✅ Anatomy of a Metaclass

Method	Purpose
__new__(mcs, name, bases, dct)	Creates and returns the new class
__init__(cls, name, bases, dct)	Optional initializer
__call__()	Controls what happens when you call the class (for advanced use)

⚙️ When to Use vs Avoid

Use When	Avoid When
Building frameworks / plugins	Simple business logic
Validating class design at creation	You can use decorators instead
Auto-registration of classes	It adds unnecessary complexity
Controlling class behaviors globally	In small/medium projects

Summary

Concept	Meaning
Metaclass	Class of a class
type	Default metaclass in Python
__new__	Called during class creation
Use Case	Frameworks, validations, auto-wiring
Real Use	Django ORM, SQLAlchemy, FastAPI, Pydantic

Happy Learning !! Thanks for Your Patience ☺

Surendra Panpaliya

GKTCS Innovations