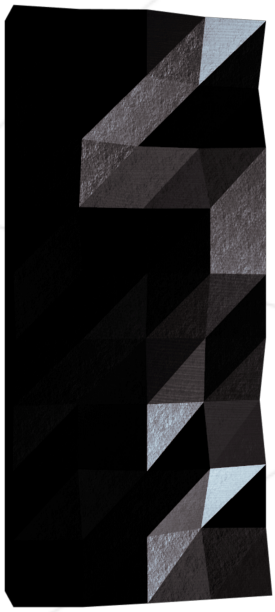

Cosmic Data

Discover Pydantic Models & Validation

Summary: Master Pydantic data validation through space-themed exercises. Learn to create robust models, implement custom validation, and handle nested structures while managing cosmic data streams.

Version: 2.0

Contents

Chapter I

Introduction

Welcome to the **Cosmic Data Observatory**, the galaxy's premier data processing facility! As a junior Data Engineer, you've been assigned to validate incoming data streams from various space missions, alien contact reports, and station monitoring systems.

Your mission: Learn **Pydantic**; Python's most powerful data validation library; through hands-on cosmic scenarios. You'll progress from basic models to complex validation rules, ensuring data integrity across the known universe.

The Cosmic Theme

Throughout this activity, you'll work with space-themed data that makes learning engaging:

- **Space Stations**: Basic data validation fundamentals
- **Alien Contacts**: Custom validation rules and logic
- **Crew Management**: Nested models and complex relationships

Each exercise builds upon the previous one, introducing new Pydantic concepts progressively.

Pro Tip: Pydantic automatically converts compatible types (like strings to datetime) and provides detailed error messages when validation fails. This makes it perfect for building robust APIs and data processing systems!

Chapter II

AI Instructions

Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

Main message

- Use AI to reduce repetitive or tedious tasks.
- Develop prompting skills — both coding and non-coding — that will benefit your future career.
- Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.
- Continue building both technical and power skills by working with your peers.
- Only use AI-generated content that you fully understand and can take responsibility for.

Learner rules:

- You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.
- You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.
- You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.
- You should always seek peer review — don't rely solely on your own validation.

Phase outcomes:

- Develop both general-purpose and domain-specific prompting skills.
- Boost your productivity with effective use of AI tools.
- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

Comments and examples:

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.
- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.
- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.
- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

✓ Good practice:

> I ask AI: "How do I test a sorting function?" It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

✗ Bad practice:

> I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

✓ Good practice:

> I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

✗ Bad practice:

> I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

Chapter III

General Instructions

III.1 Technical Requirements

- **Python 3.10 or later**
- **Code Quality**: Your code must respect the flake8 linter standards
- **Type Hints**: All functions and methods must include type hints
- **pip** package manager
- **Virtual environment** (recommended: venv, virtualenv, or conda)
- **Pydantic 2.x** (will be installed via pip)

III.2 Available Tools

This activity includes data generation tools to help you test your Pydantic models:

- **data_generator.py** - Generate realistic test data for all exercises
- **data_exporter.py** - Export data in JSON, CSV, and Python formats
- **generated_data/** - Pre-generated datasets ready to use

Authorized imports: You may import JSON and CSV data from the tools directory. Standard library modules (json, csv, datetime, etc.) are allowed.

Important: This activity focuses on **Pydantic v2** syntax. Avoid deprecated decorators like @validator - use @model_validator for custom validation instead.

III.3 Key Pydantic Concepts

BaseModel

The foundation of all Pydantic models. Inherit from `BaseModel` to create validated data classes.

Field

Use `Field(...)` to add validation constraints, descriptions, and default values to model attributes.

model_validator

Use the `@model_validator(mode='after')` decorator for custom validation logic that runs after Pydantic's built-in validation. This replaces the deprecated `@validator` decorator from Pydantic v1.

Chapter IV

Exercise 0: Space Station Data

	Exercise0
Space Station Data	
Directory: *ex*0/	
Files to Submit: `space_station.py`	
Authorized: `None`	

Objective: Learn basic Pydantic model creation with BaseModel and Field validation.

IV.1 Background

The Cosmic Data Observatory monitors hundreds of space stations across the galaxy. Each station reports vital statistics including crew size, power levels, and operational status. Your first task is to create a validation system for this critical data.

IV.2 Requirements

SpaceStation Model

Create a Pydantic model with these validated fields:

- `station_id`: String, 3-10 characters
- `name`: String, 1-50 characters
- `crew_size`: Integer, 1-20 people

- `power_level`: Float, 0.0-100.0 percent
- `oxygen_level`: Float, 0.0-100.0 percent
- `last_maintenance`: DateTime field
- `is_operational`: Boolean, defaults to True
- `notes`: Optional string, max 200 characters

Demonstration Function

Include a `main()` function that:

- Creates a valid space station instance
- Displays the station information clearly
- Attempts to create an invalid station (e.g., crew_size > 20)
- Shows the validation error message

Expected Output Example:

```
Space Station Data Validation
========================================
Valid station created:
ID: ISS001
Name: International Space Station
Crew: 6 people
Power: 85.5%
Oxygen: 92.3%
Status: Operational

========================================
Expected validation error:
Input should be less than or equal to 20
```

Think About: How does Pydantic's automatic type conversion work? What happens when you pass a string timestamp to a datetime field?

Chapter V

Exercise 1: Alien Contact Logs

	Exercise1
Alien Contact Logs	
Directory: *ex*1/	
Files to Submit: `alien_contact.py`	
Authorized: `None`	

Objective: Master custom validation using `@model_validator` for complex business rules.

V.1 Background

The Observatory receives reports of alien contact from across the galaxy. These sensitive reports require sophisticated validation rules that go beyond simple field constraints. Different contact types have different requirements, and certain combinations of data indicate potentially fraudulent reports.

V.2 Requirements

ContactType Enum

Define contact types: `radio`, `visual`, `physical`, `telepathic`

AlienContact Model

Create a Pydantic model with these fields:

- `contact_id`: String, 5-15 characters
- `timestamp`: DateTime of contact
- `location`: String, 3-100 characters
- `contact_type`: ContactType enum
- `signal_strength`: Float, 0.0-10.0 scale
- `duration_minutes`: Integer, 1-1440 (max 24 hours)
- `witness_count`: Integer, 1-100 people
- `message_received`: Optional string, max 500 characters
- `is_verified`: Boolean, defaults to False

Custom Validation Rules

Implement `@model_validator(mode='after')` with these business rules:

- Contact ID must start with "AC" (Alien Contact)
- Physical contact reports must be verified
- Telepathic contact requires at least 3 witnesses
- Strong signals (> 7.0) should include received messages

Demonstration Function

Show valid and invalid contact reports with clear error messages.

Expected Output Example:

```
Alien Contact Log Validation
========================================
Valid contact report:
ID: AC_2024_001
Type: radio
Location: Area 51, Nevada
Signal: 8.5/10
Duration: 45 minutes
Witnesses: 5
Message: 'Greetings from Zeta Reticuli'

========================================
Expected validation error:
Telepathic contact requires at least 3 witnesses
```

Advanced Tip: The @model_validator decorator allows you to validate the entire model after all fields have been validated. Remember to return self at the end of your validator function.

Chapter VI

Exercise 2: Space Crew Management

	Exercise2
Space Crew Management	
Directory: *ex2/*	
Files to Submit: `space_crew.py`	
Authorized: `None`	

Objective: Master nested Pydantic models and complex data relationships.

VI.1 Background

Space missions require careful crew management. Each mission has multiple crew members with different ranks, specializations, and experience levels. The Observatory needs to validate that mission crews meet safety and operational requirements before launch approval.

VI.2 Requirements

Rank Enum

Define crew ranks: `cadet`, `officer`, `lieutenant`, `captain`, `commander`

CrewMember Model

Individual crew member with these fields:

- `member_id`: String, 3-10 characters
- `name`: String, 2-50 characters
- `rank`: Rank enum
- `age`: Integer, 18-80 years
- `specialization`: String, 3-30 characters
- `years_experience`: Integer, 0-50 years
- `is_active`: Boolean, defaults to True

SpaceMission Model

Mission with crew list and these fields:

- `mission_id`: String, 5-15 characters
- `mission_name`: String, 3-100 characters
- `destination`: String, 3-50 characters
- `launch_date`: DateTime
- `duration_days`: Integer, 1-3650 days (max 10 years)
- `crew`: List of CrewMember, 1-12 members
- `mission_status`: String, defaults to "planned"
- `budget_millions`: Float, 1.0-10000.0 million dollars

Mission Validation Rules

Implement `@model_validator(mode='after')` with these safety requirements:

- Mission ID must start with "M"
- Must have at least one Commander or Captain
- Long missions (> 365 days) need 50% experienced crew (5+ years)
- All crew members must be active

Demonstration Function

Show a valid mission with crew details and an invalid mission that fails validation.

Expected Output Example:

```
Space Mission Crew Validation
========================================
Valid mission created:
Mission: Mars Colony Establishment
ID: M2024_MARS
Destination: Mars
Duration: 900 days
Budget: $2500.0M
Crew size: 3
Crew members:
- Sarah Connor (commander) - Mission Command
- John Smith (lieutenant) - Navigation
- Alice Johnson (officer) - Engineering

========================================
Expected validation error:
Mission must have at least one Commander or Captain
```

Think About: How does Pydantic handle validation of nested models? What happens when a CrewMember fails validation within a SpaceMission?

Chapter VII

Submission and peer-review

Submit your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

`You need to return only the files requested by the subject of this activity.`