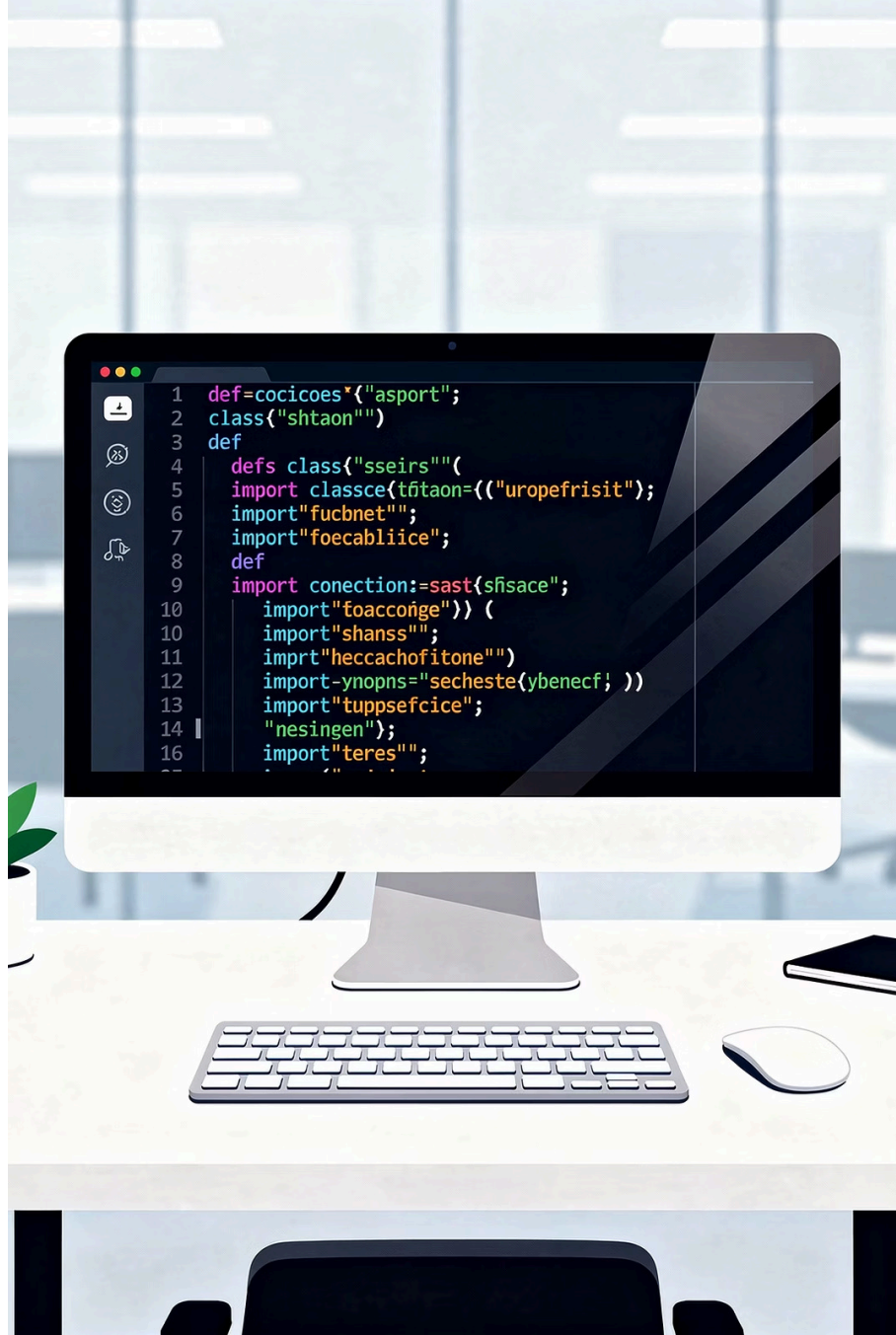
Functions in Python

Parameters, Return Values, and Building Reusable Code



Learning Goals

By the end of this module, participants will be able to:

- Define and use functions with parameters and return values
- Distinguish between printing and returning data in functions
- Read and write CSV files using Python's csv module
- Handle runtime errors gracefully using try/except blocks
- Build fault-tolerant data processing pipelines
- Apply these concepts in real Data Engineering scenarios

Module Structure

01

Functions in Python

Understanding parameters, return values, and the difference between printing and returning data

02

File Handling & CSV Processing

Reading and writing CSV files using csv.DictReader and csv.DictWriter

03

Error Handling

Using try/except blocks to build resilient data pipelines

Lesson 1: Functions in Python

Objective: By the end of this lesson, participants will understand how to define functions with parameters, return values from functions, and distinguish between printing and returning data.

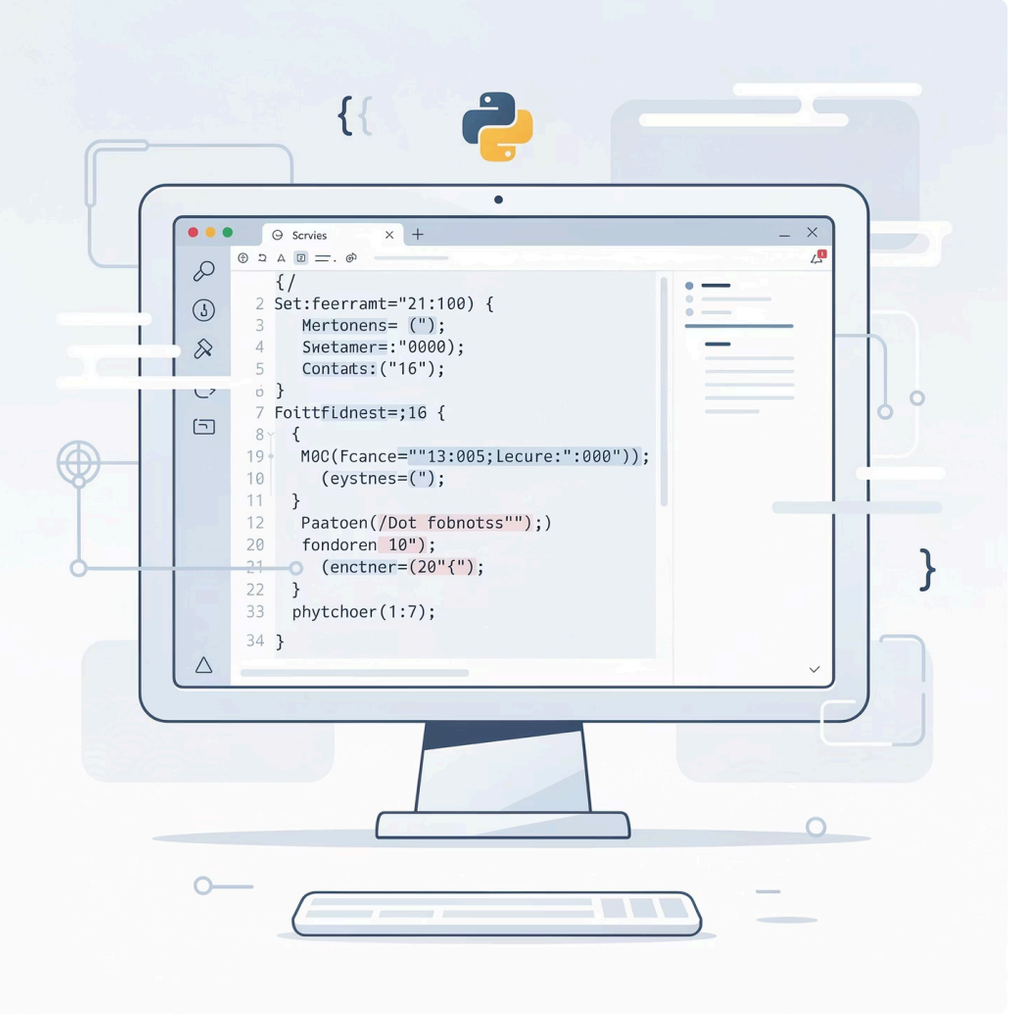
Key Concepts: What Are Functions?

Definition

Functions are reusable blocks of code that accept input (parameters), perform specific tasks, and optionally return results.

Why Functions Matter

- Avoid code duplication
- Improve code organization
- Enable testing and debugging
- Make pipelines maintainable



Trainer Tip: Ask participants: "Where have you seen repeated logic in your code?"

Parameters: Making Functions Flexible

What Are Parameters?

Variables that allow functions to accept different inputs each time they're called

Why Use Them?

Enable the same function to process different data without rewriting code

In Data Pipelines

Validation functions can check any record, transformation functions can clean any dataset

Example: A function `validate_age(age)` can check any age value passed to it.

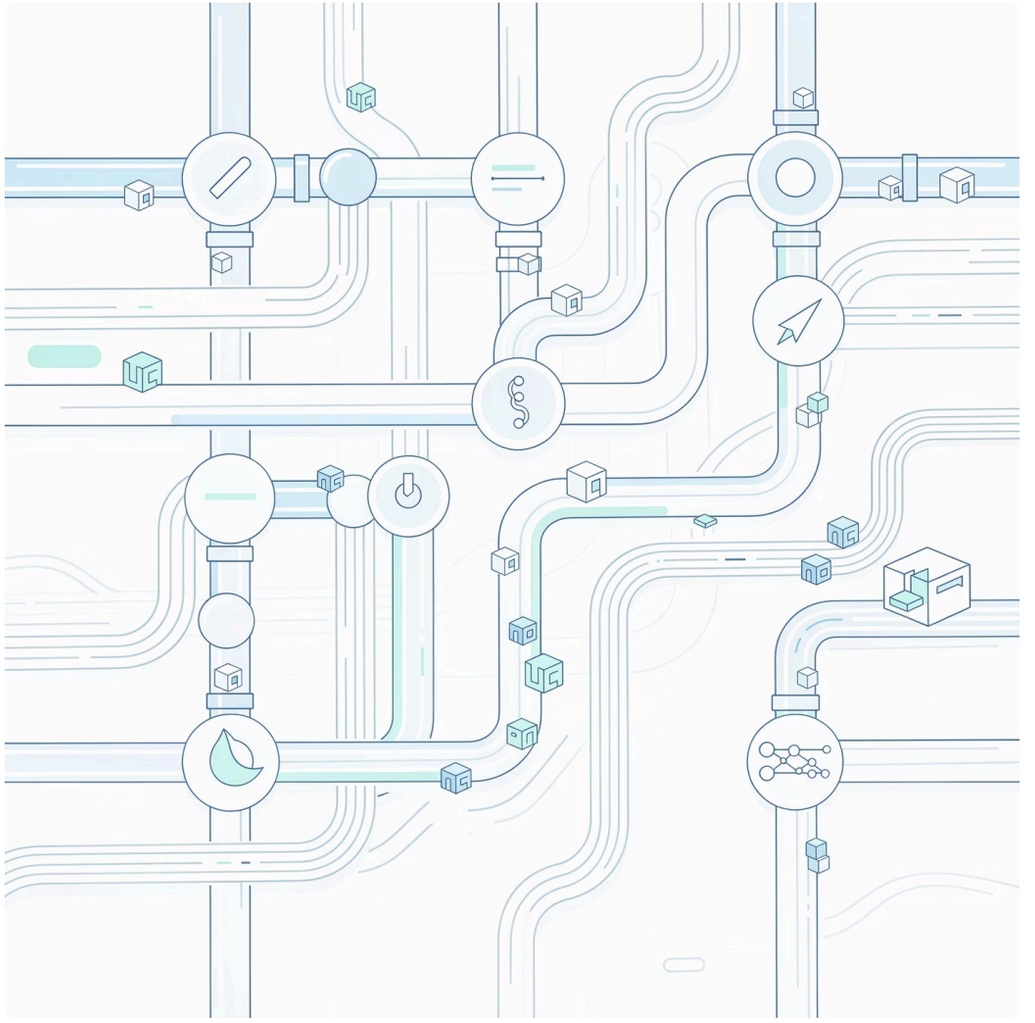
Return Values: Getting Results Back

What is a Return Value?

The output that a function sends back to the caller, allowing the result to be stored, reused, or passed to another function.

Key Differences

- `print()` displays data on screen
- `return` sends data back to the program
- Only return values can be stored in variables
- Only return values can be used in pipelines



Trainer Tip: Whiteboard example showing `result = add(5, 3)` vs `print(5 + 3)`

Printing vs Returning: Critical Distinction

print() Function

- Displays data to console
- Returns None
- Cannot store result
- Use for debugging only

return Statement

- Sends data back to caller
- Result can be stored
- Result can be reused
- Essential for pipelines

In production data engineering, functions almost always return values. Printing is for debugging.

Real-World Perspective: Functions in Data Engineering

When Used in Production

- Data validation rules applied consistently across datasets
- Transformation logic reused in multiple pipelines
- Calculation functions that process batches of records
- ETL steps broken into testable units

Typical Tools Involved

- Apache Airflow tasks (each task is essentially a function)
- AWS Lambda functions
- dbt models (SQL as functions)
- Custom Python data processing scripts

Trainer Tip: Connect to previous lessons on data types and control flow

Common Patterns in Data Pipelines

1

Validation Functions

Check data quality, return True/False

2

Transformation Functions

Clean and standardize data, return processed values

3

Calculation Functions

Aggregate or compute metrics, return numeric results

4

Chained Functions

Output of one becomes input of next (ETL pattern)

Common Misconceptions

Misconception #1

"If I see output, the function must be working correctly"

~~Wrong:~~ Printing and returning are fundamentally different

Misconception #2

"Functions are only for avoiding repeated code"

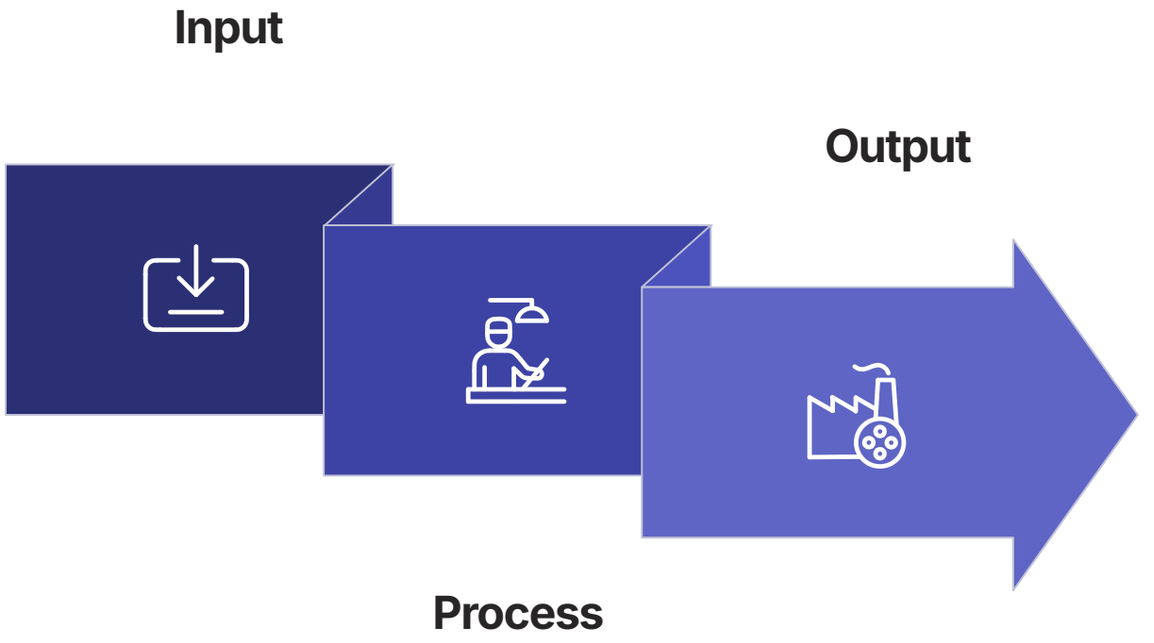
~~Wrong:~~ Functions also improve testability, readability, and maintainability

Misconception #3

"I can put print statements anywhere and it won't matter"

~~Wrong:~~ Print statements in production logs can cause performance issues

Visual: Function Flow



This is the fundamental pattern for every function in data processing pipelines.

Trainer Tip: Ask participants to identify the input, process, and output in their recent code

Activity 1 – Write Reusable Utility Functions

In this hands-on activity, participants will convert repeated validation logic into small, reusable utility functions that accept parameters and return values.

What Participants Will Achieve

- Write functions that validate data using parameters
- Use return values instead of print statements
- Combine multiple validation functions together
- Apply function-based validation to a list of student records

Expected Time: 25-30 minutes

Activity 1: Skills Developed

Function Design

Create focused functions that do one thing well

Code Reusability

Write logic once, use it many times

Function Composition

Combine small functions to build larger capabilities

Pipeline Thinking

Understand how functions form the backbone of data processing

This activity reinforces the core principle: **functions with return values are the building blocks of production pipelines.**

Lesson 2: File Handling and CSV Processing

Objective: By the end of this lesson, participants will understand how to read data from CSV files using csv.DictReader, write data to CSV files using csv.DictWriter, and apply these skills in ETL workflows.

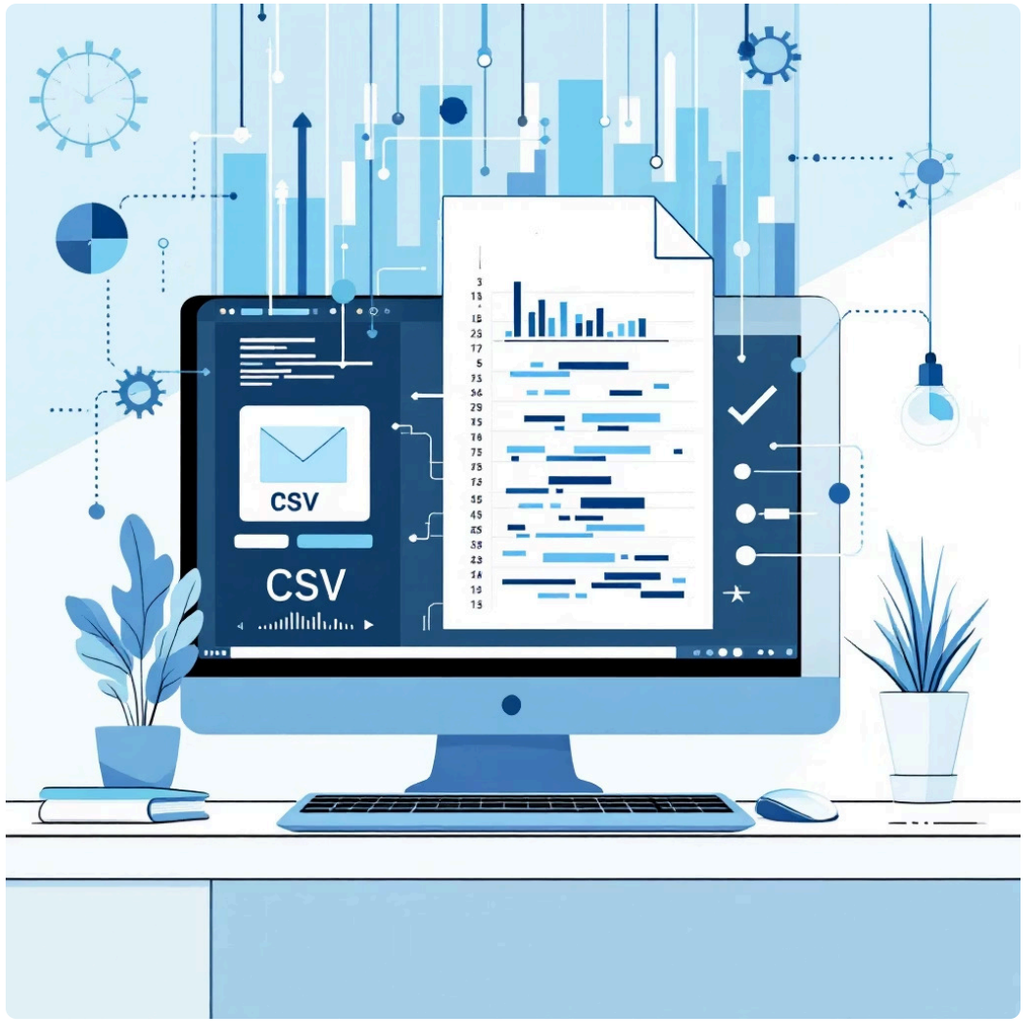
Key Concepts: Why CSV Files?

What Are CSV Files?

CSV (Comma-Separated Values) files store tabular data in plain text format, with each row representing a record and columns separated by commas.

Why CSV Matters

- Universal format supported by all tools
- Human-readable and simple
- Common data exchange format
- Used extensively in ETL pipelines



Trainer Tip: Ask participants: "What file formats have you used to share data?"

csv.DictReader: Reading CSV as Dictionaries

What It Does

Reads each row of a CSV file as a Python dictionary, using column headers as keys

Why Use It?

Access fields by name (row["age"]) instead of index (row[2]), making code self-documenting

In Data Pipelines

Standard way to ingest CSV data in Extract phase of ETL workflows

Key advantage: When column order changes, code using DictReader still works correctly.

csv.DictWriter: Writing Structured Data

What It Does

Writes Python dictionaries to CSV files, ensuring consistent column structure and automatically handling headers.

Important Parameters

- `fieldnames`: list of column names
- `writeheader()`: writes column names
- `writerow()`: writes single dictionary
- `writerows()`: writes list of dictionaries

Trainer Tip: Emphasize the importance of `newline=""` parameter on Windows

The ETL Pattern with CSV Files

1

Extract

Read data from CSV using csv.DictReader, each row becomes a dictionary

2

Transform

Apply validation, cleaning, or calculations to dictionary values

3

Load

Write processed dictionaries to new CSV using csv.DictWriter

This is the fundamental workflow for file-based data processing.

Real-World Perspective: CSV in Data Engineering

When Used in Production

- Data exports from databases or APIs
- Data exchange between systems and teams
- Batch processing workflows
- Data ingestion into data warehouses
- Intermediate storage in multi-step pipelines

Typical Tools Involved

- Airflow tasks reading/writing CSV files
- Pandas DataFrames (built on similar concepts)
- AWS S3 buckets storing CSV data
- Snowflake COPY commands loading CSV data

Trainer Tip: Whiteboard a simple ETL flow with CSV as intermediate format

Important: Data Type Handling

1

All CSV Values Are Strings

Even numbers like "42" or "3.14" are read as text

2

Explicit Conversion Required

Use int(), float(), or other converters after reading

3

Handle Conversion Errors

Invalid values like "N/A" will cause errors if not handled

This is why error handling (covered in Lesson 3) is essential when processing CSV files.

Common Patterns: Filter and Write

One of the most common patterns in data processing: read a CSV file, filter records based on criteria, and write valid records to a new file.

01

Open Input File

Use csv.DictReader to read records

02

Apply Filter Logic

Check each record against validation rules

03

Collect Valid Records

Store passing records in a list

04

Write Output File

Use csv.DictWriter to create clean dataset

Common Misconceptions

Misconception #1

"CSV files are outdated, nobody uses them anymore"

~~Wrong:~~ CSV remains one of the most common data exchange formats in enterprise systems

Misconception #2

"I can treat CSV values as numbers directly"

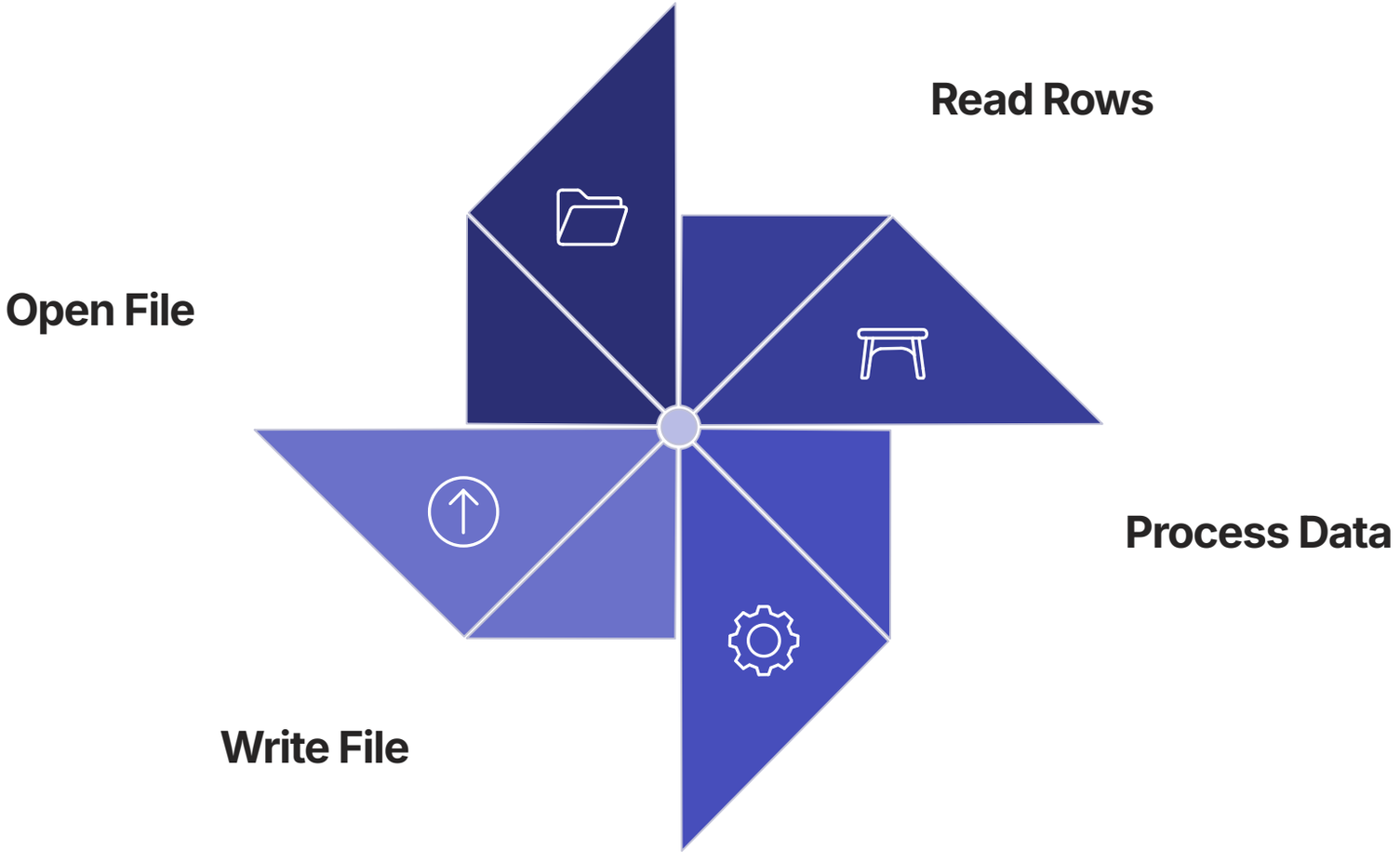
~~Wrong:~~ All CSV values are strings and must be explicitly converted

Misconception #3

"DictReader is slower than regular reader"

~~Wrong:~~ The readability and safety benefits far outweigh minimal performance differences

Visual: CSV Processing Workflow



This pattern scales from small files to large batch processing jobs.

Trainer Tip: Ask participants to think about where validation should happen in this flow

Activity 2 – Read CSV and Compute Summary Statistics

In this hands-on activity, participants will read student data from a CSV file, convert string values to numbers, and calculate summary statistics like average, minimum, and maximum scores.

What Participants Will Achieve

- Read CSV data using csv.DictReader
- Convert string values to appropriate numeric types
- Compute total, count, average, min, and max values
- Transform raw file data into meaningful metrics

Expected Time: 20-25 minutes

Activity 2: Skills Developed

File Reading

Extract data from external files safely

Type Conversion

Handle the reality that CSV values are always strings

Statistical Aggregation

Compute meaningful metrics from raw data

Extract-Transform Pattern

Foundation for more complex ETL workflows

This activity demonstrates how raw files become actionable insights through data processing.

Lesson 3: Error Handling in Python

Objective: By the end of this lesson, participants will understand how to use try/except blocks to handle runtime errors gracefully, prevent pipeline crashes, and build fault-tolerant data processing systems.

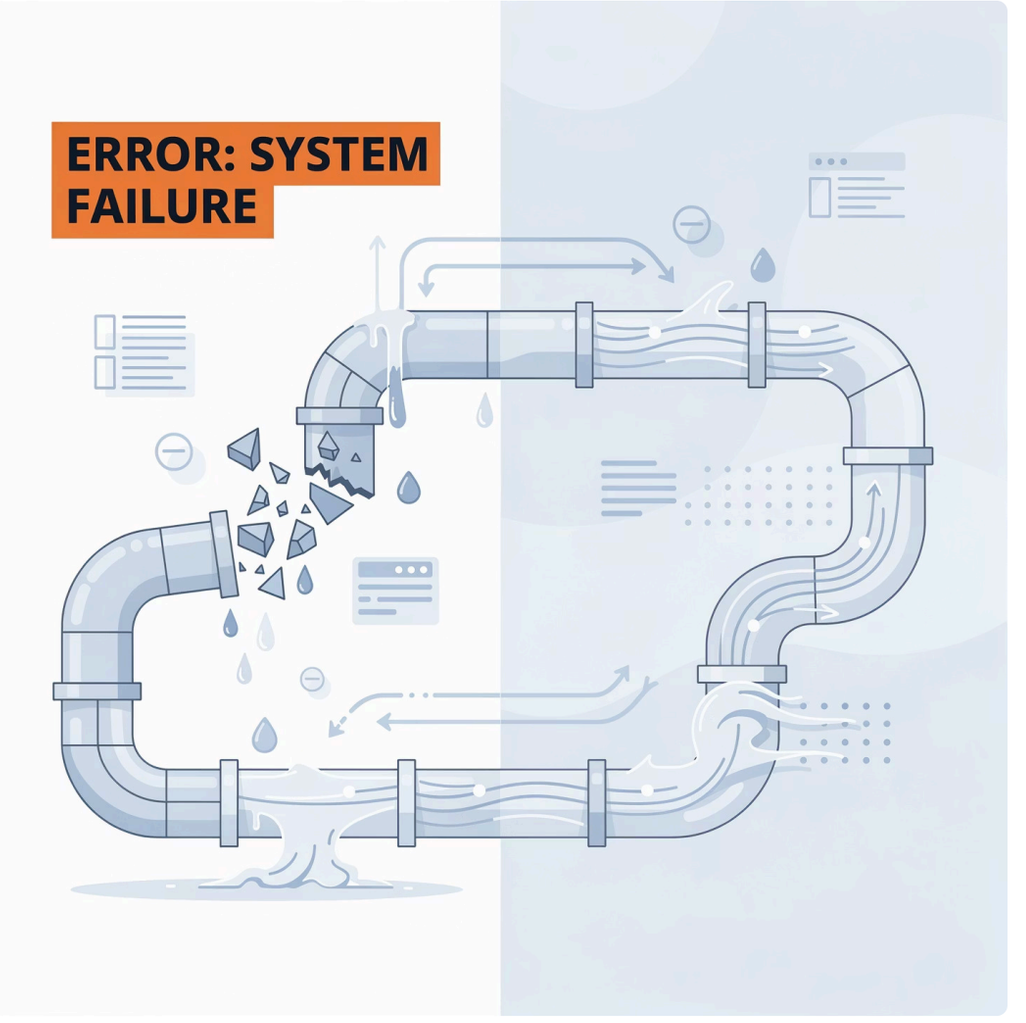
Key Concepts: What Are Exceptions?

Definition

Exceptions are runtime errors that interrupt the normal flow of a program. Without handling, they cause the program to crash.

Why Error Handling Matters

- Prevents entire pipelines from crashing
- Allows graceful degradation
- Enables logging and debugging
- Separates valid from invalid data



Trainer Tip: Ask participants: "What happens when one bad record crashes your pipeline?"

try / except: Controlled Error Handling

try Block

Contains code that might raise an exception

except Block

Handles the error if it occurs, preventing crash

Result

Program continues running, invalid data is handled separately

The difference between code that crashes and code that handles errors gracefully.

Common Python Exceptions

ValueError

Invalid data conversion (e.g., int("abc"))

KeyError

Missing dictionary key

TypeError

Invalid operation on data type

FileNotFoundError

Attempting to open non-existent file

IndexError

Invalid list index access

ZeroDivisionError

Division by zero

Best Practice: Always catch specific exceptions, not bare `except:`

Real-World Perspective: Error Handling in Data Engineering

When Used in Production

- Processing files with inconsistent data quality
- Ingesting data from external APIs (network failures)
- Batch jobs that should complete despite individual record errors
- Data validation in ETL pipelines
- Graceful degradation when external systems are unavailable

Typical Tools Involved

- Airflow task retries and error callbacks
- AWS Lambda error handling and dead-letter queues
- Data quality monitoring tools (Great Expectations, deequ)
- Centralized logging systems (CloudWatch, Datadog)

Trainer Tip: Connect to previous lessons: combining functions, CSV reading, and error handling

Best Practices for Exception Handling

1

Catch Specific Exceptions

Use `except ValueError:` instead of bare `except:`

2

Log Error Details

Use `except ValueError as e:` to capture error message

3

Separate Valid from Invalid

Keep good records moving, flag bad ones for review

4

Avoid Silent Failures

Never use `except: pass` in production code

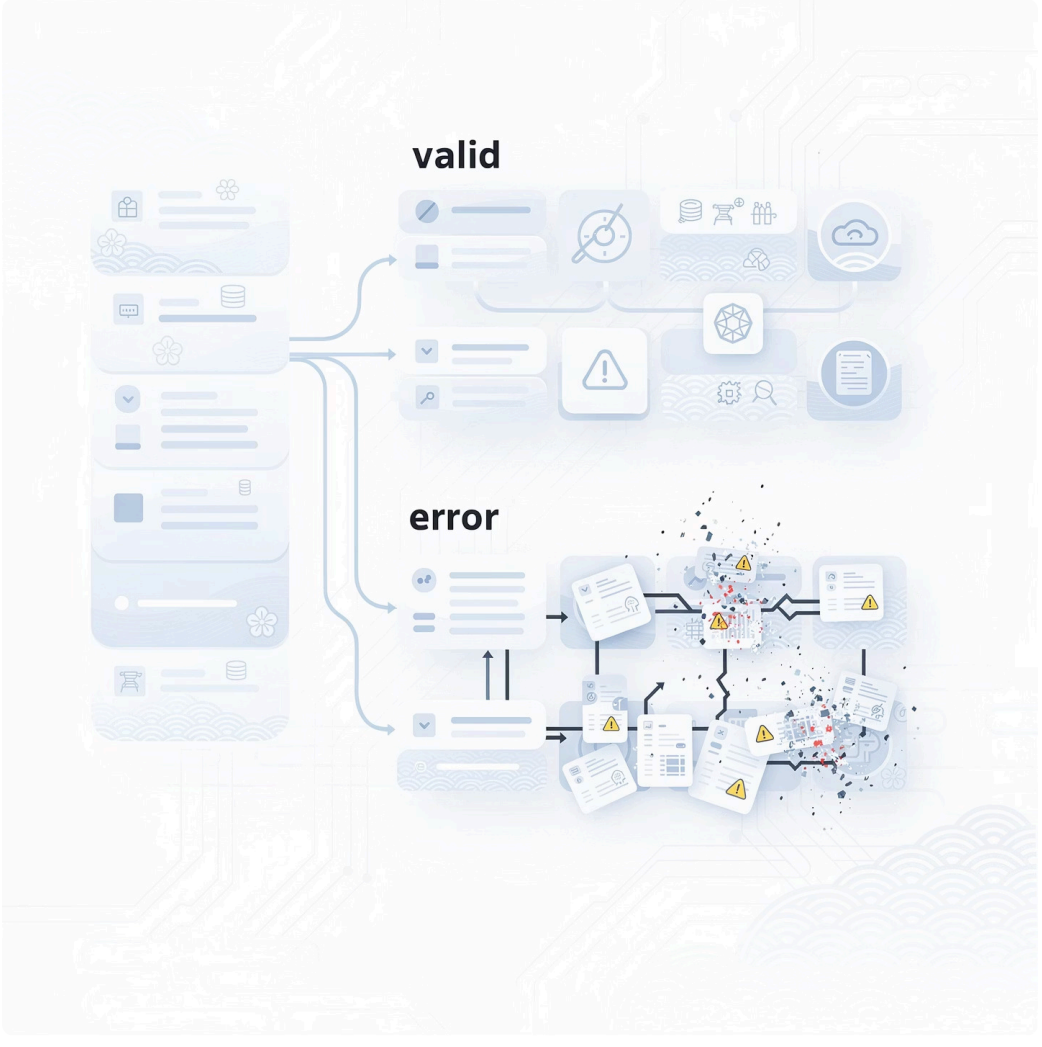
Pattern: Fault-Tolerant Record Processing

The Pattern

1. Initialize separate lists for valid and invalid records
2. Loop through input data
3. Wrap risky operations in try block
4. Catch exceptions and store problem records
5. Continue processing remaining records

Why This Works

One bad record doesn't break the entire pipeline. Invalid records are isolated for investigation.



else and finally Clauses

else Clause

Runs only if no exception occurred in try block

finally Clause

Always runs, regardless of exceptions (used for cleanup)

Common Use

Closing files, database connections, or releasing resources

Example: Use `finally` to ensure a file is closed even if an error occurs during reading.

Common Misconceptions

Misconception #1

"If my code doesn't have errors now, I don't need try/except"

~~Wrong:~~ Real-world data is unpredictable. Production code must handle unexpected inputs.

Misconception #2

"Using except: without specifying exception type is fine"

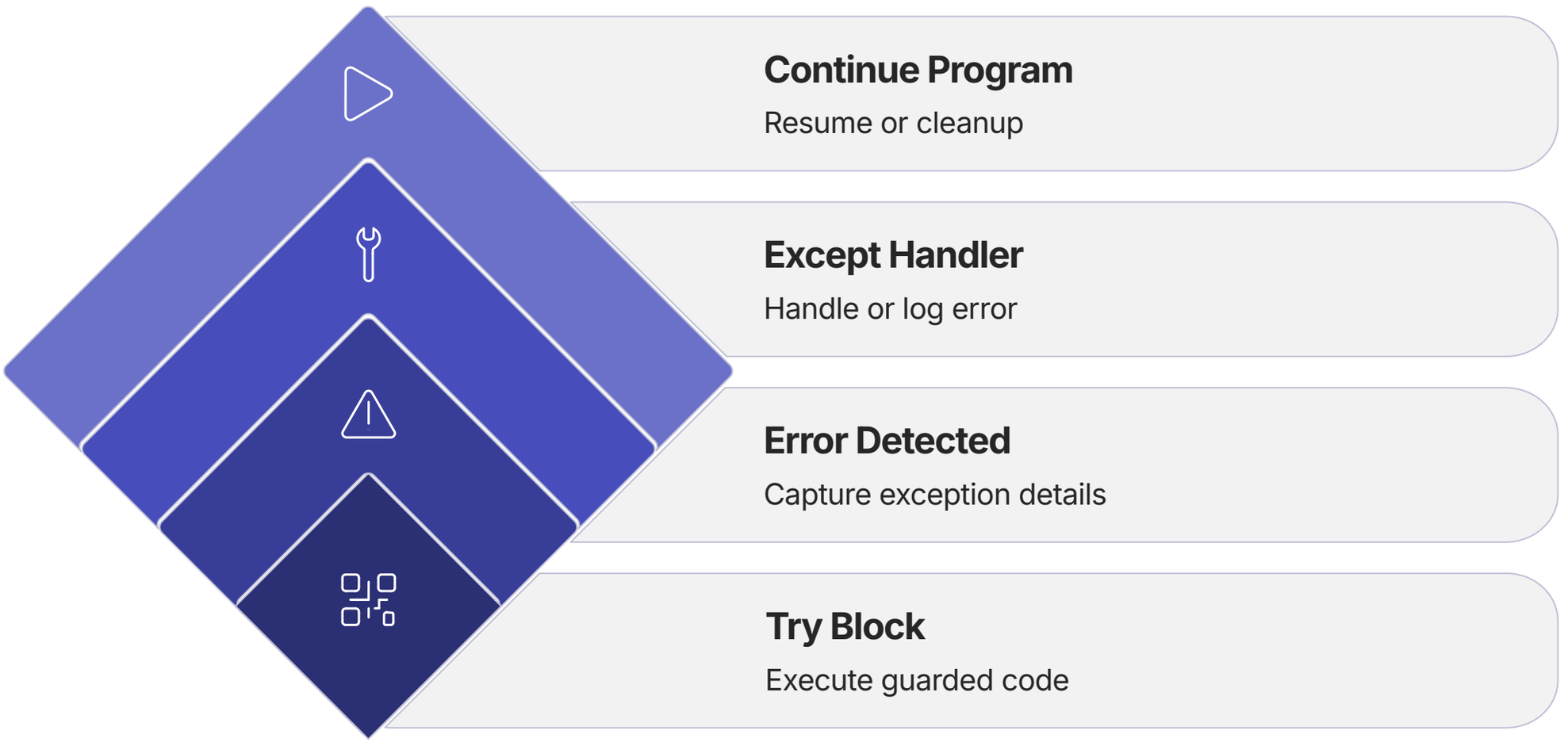
~~Wrong:~~ This catches everything, including keyboard interrupts, making debugging impossible

Misconception #3

"Error handling slows down my code"

~~Wrong:~~ The cost is negligible compared to pipeline crashes and data loss

Visual: Error Handling Flow



This pattern is the difference between brittle scripts and production-ready systems.

Trainer Tip: Ask participants to identify where error handling should be added in their previous activities

Activity 3 – Make a CSV Reader Fault-Tolerant

In this hands-on activity, participants will take a CSV file containing intentional data quality issues and build a fault-tolerant reader that handles errors gracefully without crashing.

What Participants Will Achieve

- Use try/except blocks around risky data operations
- Separate valid records from invalid records
- Attach error messages to problematic records
- Ensure the pipeline completes even when errors occur

Expected Time: 25-30 minutes

Activity 3: Skills Developed

Defensive Programming

Anticipate and handle failures before they crash your code

Data Quality Separation

Isolate problematic records for review while processing continues

Error Logging

Capture meaningful error details for debugging

Production Readiness

Build pipelines that handle real-world messy data

This activity ties together functions, CSV processing, and error handling into a complete, production-ready workflow.

Bringing It All Together

1

Functions

Organize logic into reusable, testable units

2

CSV Processing

Read and write structured data files

3

Error Handling

Prevent crashes and handle bad data gracefully

4

Result

Production-ready data processing pipelines

These three concepts form the foundation of every data engineering workflow.

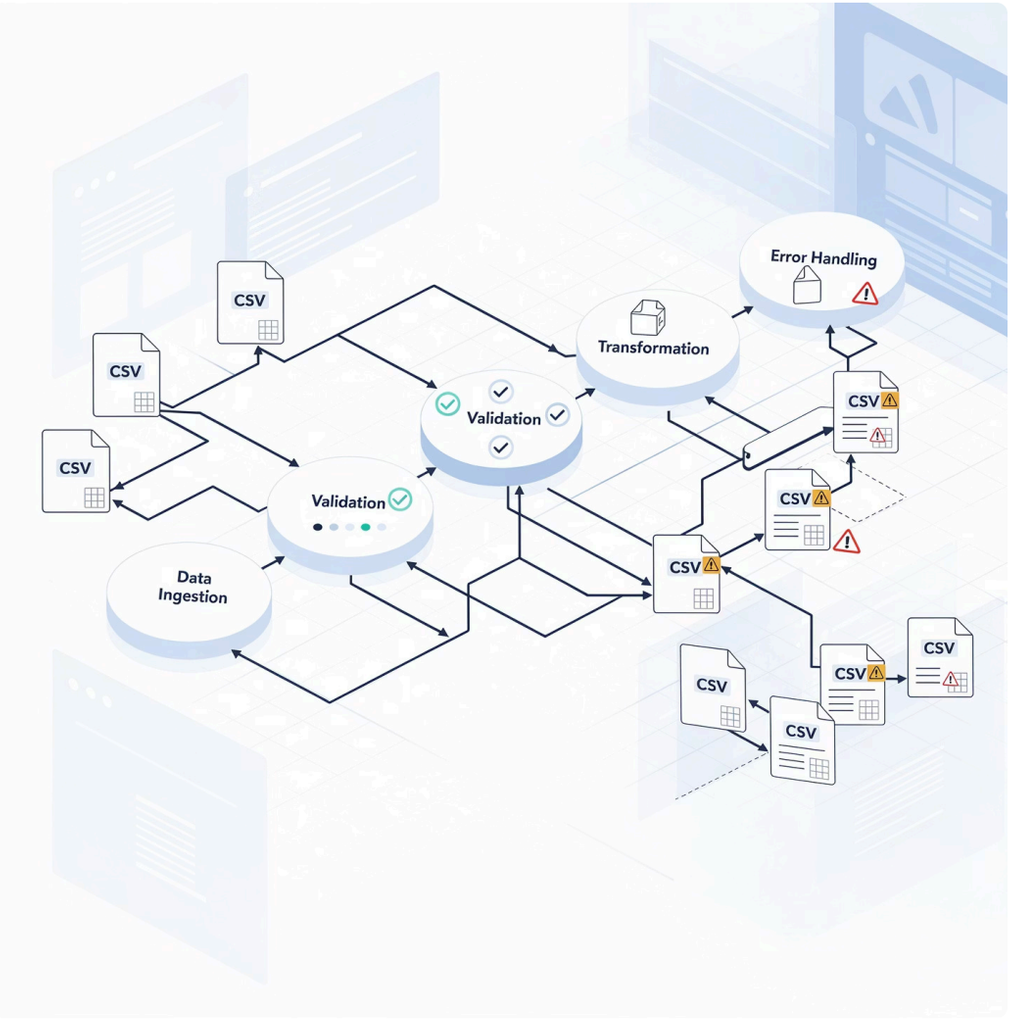
Real Production Pipeline Example

Scenario

Processing daily sales reports from multiple stores, where files may have inconsistent formats or missing values.

Applied Concepts

- **Functions:** Validation rules, transformation logic
- **CSV:** Read raw exports, write clean datasets
- **Errors:** Handle format issues, missing files



Trainer Tip: Walk through how each concept applies at different pipeline stages

Key Takeaways: Functions

Functions are the building blocks of data pipelines

Each function should have a clear, single purpose

Parameters make functions flexible and reusable

Same logic can process different data

Return values enable function chaining

Output of one becomes input of next (ETL pattern)

Print for debugging, return for production

Only return values can be stored and reused

Key Takeaways: CSV Processing

CSV is a universal data exchange format

Supported by every tool, simple and portable

DictReader makes code self-documenting

Access fields by name, not index

All CSV values are strings

Explicit type conversion is always required

CSV processing follows the ETL pattern

Extract from file, transform data, load to new file

Key Takeaways: Error Handling

Production code must handle unexpected inputs

Real-world data is messy and unpredictable

Catch specific exceptions, not all exceptions

Precise error handling enables better debugging

Separate valid records from invalid records

Don't let one bad record crash the entire pipeline

Log error details for investigation

Silent failures are worse than crashes

Common Pitfalls to Avoid

Functions

- Forgetting to return values
- Using print instead of return
- Writing functions that do too many things

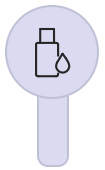
CSV Processing

- Assuming values are already numbers
- Not handling missing or empty fields
- Forgetting newline="" on Windows

Error Handling

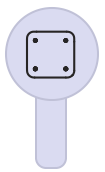
- Using bare except:
- Silent failures with pass
- Not logging error details

Skills Progression



Foundation

Variables, data types, control flow

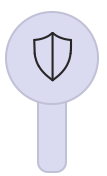


Building Blocks

Functions, parameters, return values

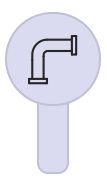
Data Access

File handling, CSV processing



Resilience

Error handling, fault tolerance



Production Ready

Complete ETL workflows

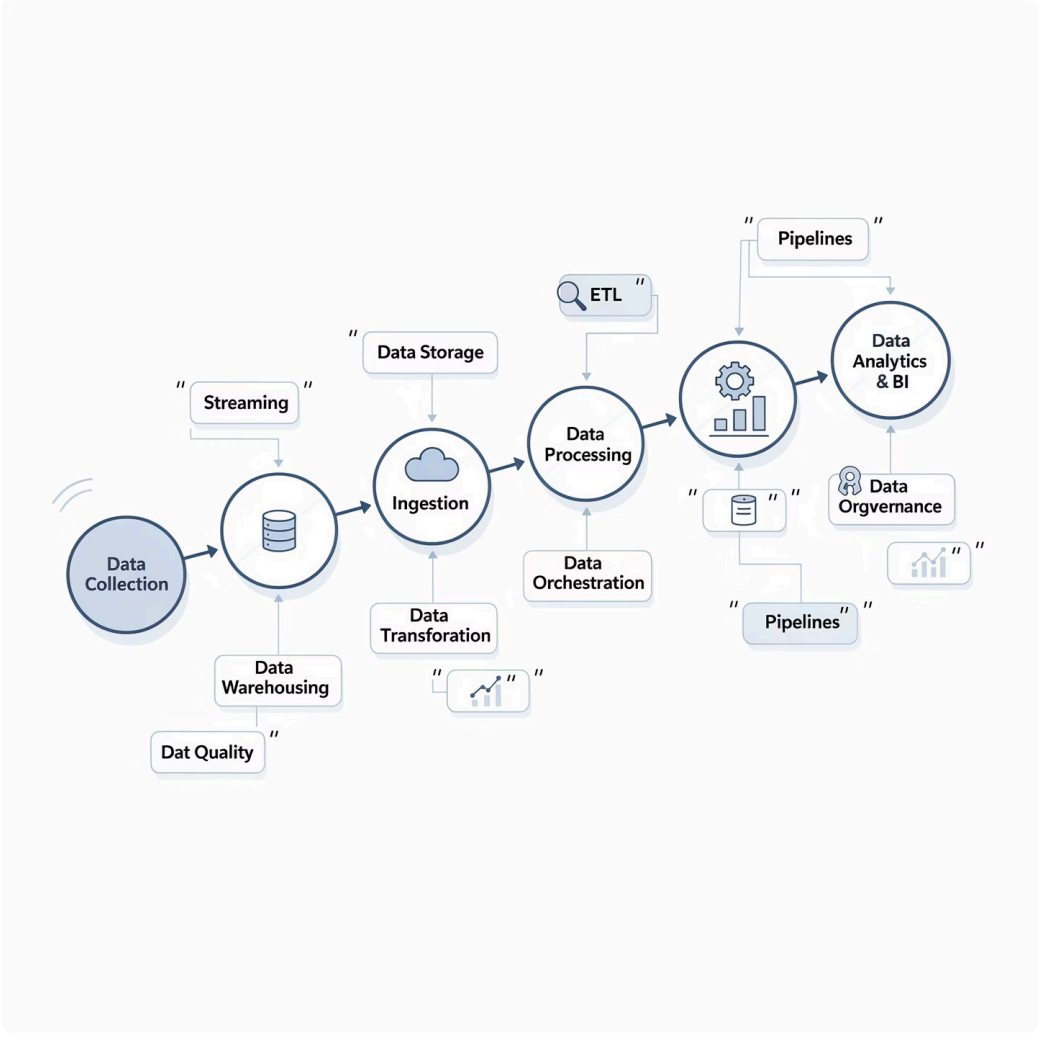
Next Steps in Your Learning Journey

What You've Mastered

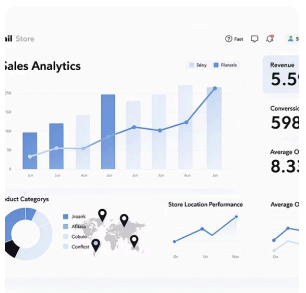
- Writing reusable functions
- Processing CSV files
- Handling errors gracefully
- Building basic ETL workflows

What's Coming Next

- Working with databases
- Advanced data transformations
- Working with APIs
- Scheduling and orchestration



Practical Applications in Industry



Retail Analytics

Processing daily sales files from stores, validating transactions, computing metrics

Financial Services

Ingesting transaction logs, detecting anomalies, generating compliance reports

Healthcare

Processing patient records, ensuring data quality, maintaining HIPAA compliance

E-commerce

Analyzing customer behavior, processing order data, building recommendation systems

Tools and Technologies

The concepts you've learned form the foundation for working with professional data engineering tools.

Apache Airflow

Tasks are Python functions; DAGs orchestrate workflows

AWS Lambda

Serverless functions triggered by events

Pandas

Built on the same CSV reading concepts, scaled up

PySpark

Distributed data processing using Python functions

dbt

SQL functions for data transformation

Great Expectations

Data validation framework using similar error handling patterns

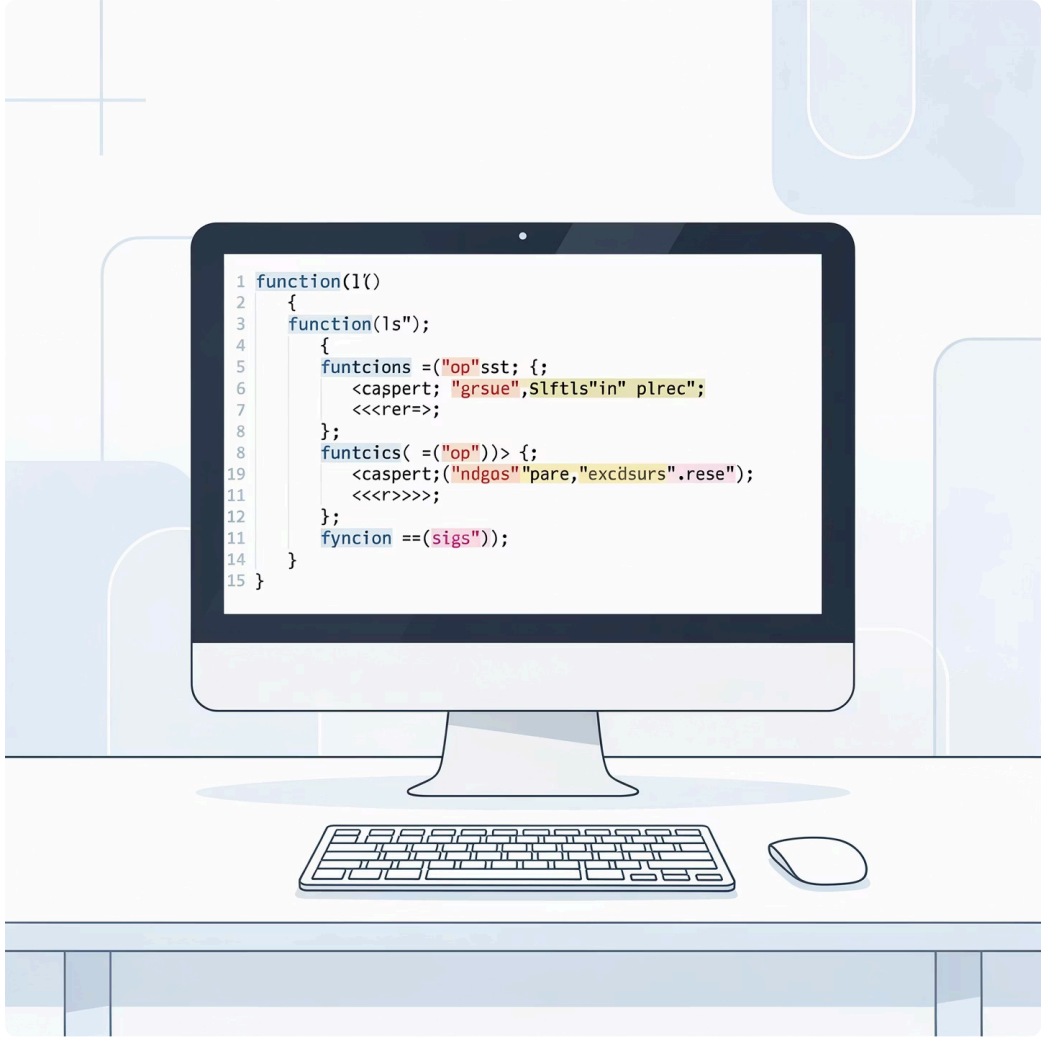
Code Quality Matters

Why Clean Code Is Critical

- Easier to debug when issues arise
- Other engineers can understand and maintain it
- Reduces time spent fixing production incidents
- Enables team collaboration

Best Practices

- Use descriptive function and variable names
- Write docstrings for complex functions
- Keep functions focused and small
- Handle errors explicitly
- Add comments for non-obvious logic



Testing Your Code

Functions with clear parameters and return values are easy to test.

Unit Testing

Test individual functions in isolation with known inputs and expected outputs

Integration Testing

Test that functions work correctly together in a pipeline

Data Quality Testing

Verify validation rules catch expected errors

Trainer Tip: Briefly mention pytest or unittest as next learning topics

Performance Considerations

When Performance Matters

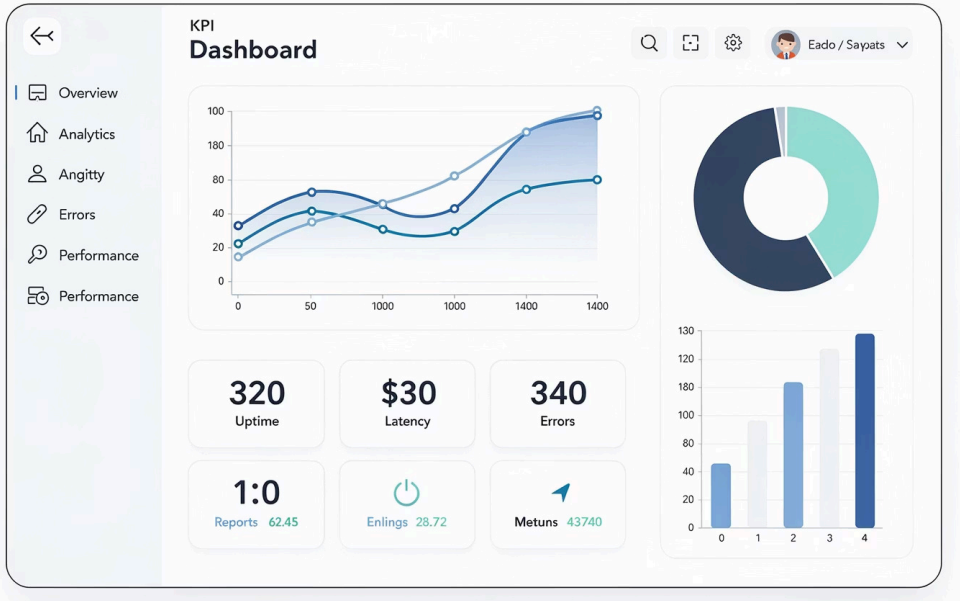
- Processing files with millions of rows
- Real-time or near-real-time pipelines
- Limited compute resources

Optimization Strategies

- Process data in batches
- Use generators for large files
- Minimize redundant conversions
- Profile code to find bottlenecks

Remember

Start with correct, readable code. Optimize only when necessary.



Debugging Strategies

01

Read the Error Message

Python error messages tell you exactly what went wrong and where

02

Add Print Statements

Inspect variable values at different points in execution

03

Use a Debugger

Step through code line by line to understand flow

04

Isolate the Problem

Test functions individually with known inputs

05

Check Assumptions

Verify data types, file paths, and edge cases

Documentation and Resources

Official Python Documentation

Comprehensive reference for all Python features and standard library modules

Real Python

In-depth tutorials and practical examples for Python developers

Stack Overflow

Community-driven Q&A for specific problems and solutions

Python Tutor

Visualize code execution step by step

Practice Recommendations

Reinforce Your Learning

1. Redo the activities from scratch without looking at solutions
2. Modify activities with new requirements
3. Find a real CSV dataset and process it
4. Build a small project combining all concepts

Project Ideas

- Sales data analyzer
- Student grade calculator
- CSV file merger/splitter
- Data quality report generator

From Scripts to Systems

1

Scripts

One-time data processing tasks

2

Reusable Functions

Modular logic that can be tested

3

Robust Pipelines

Error handling and data quality checks

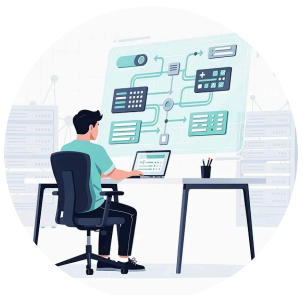
4

Production Systems

Scheduled, monitored, production-grade workflows

You've completed the first three stages. The final stage builds on everything you've learned.

Career Paths in Data Engineering

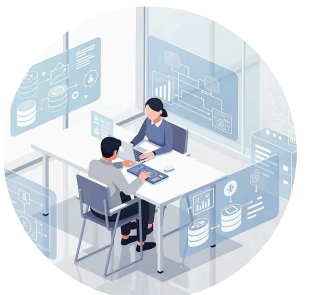


Data Engineer

Build and maintain data pipelines, ETL workflows, data warehouses

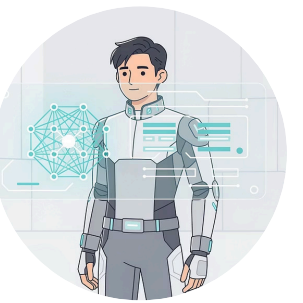
Analytics Engineer

Transform data for analysis, build metrics, create dashboards



Data Architect

Design data systems, choose technologies, establish standards



ML Engineer

Prepare data for machine learning, deploy models, monitor performance

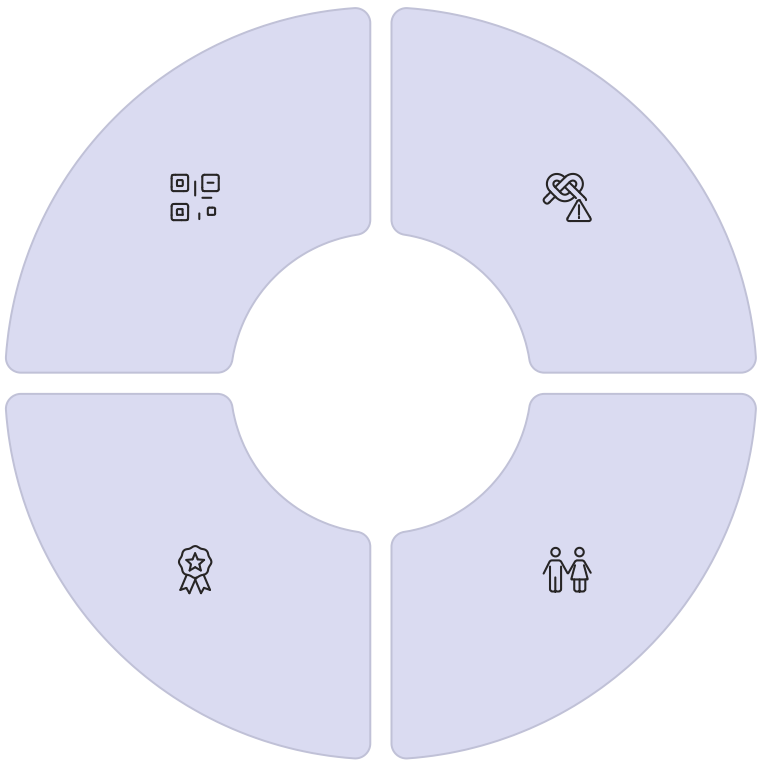
What Makes a Great Data Engineer

Technical Skills

- Python, SQL
- Pipeline tools
- Cloud platforms

Problem Solving

- Debugging complex issues
- Optimizing performance
- Designing solutions



Quality Focus

- Data accuracy
- System reliability
- Error handling

Communication

- Working with stakeholders
- Documenting systems
- Collaborating with teams

Continuous Learning

Technology Evolves Rapidly

Data engineering tools and best practices change frequently. Successful data engineers commit to continuous learning.

Stay Current By

- Following industry blogs and newsletters
- Taking online courses and certifications
- Contributing to open source projects
- Attending conferences and meetups
- Experimenting with new tools

Module Review

Functions

Parameters, return values, reusable logic, function composition

CSV Processing

DictReader, DictWriter, ETL pattern, type conversion

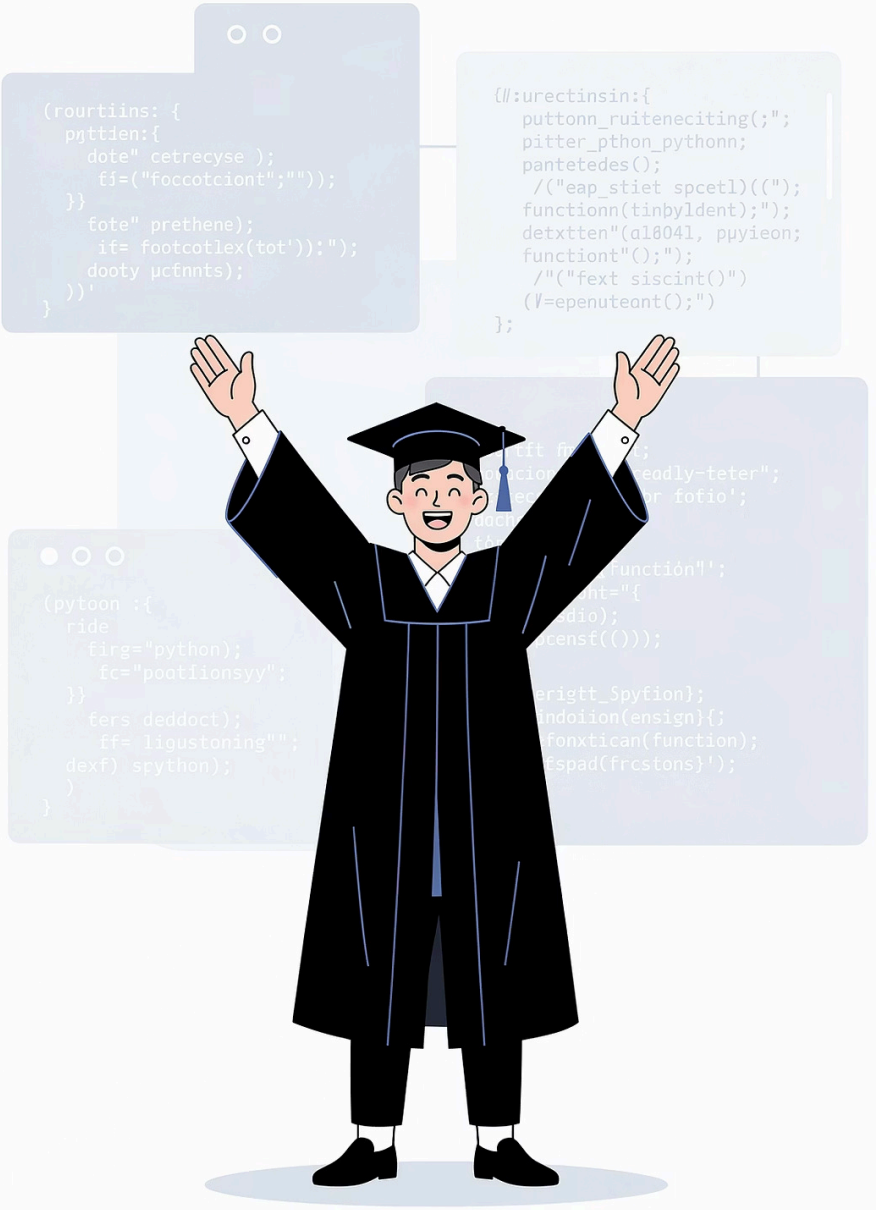
Error Handling

try/except, specific exceptions, fault tolerance, logging

Integration

Building complete, production-ready data processing pipelines

These concepts work together to form the foundation of professional data engineering.



You're Ready for Production

You now have the core skills to build real-world data processing systems. Continue practicing, keep learning, and apply these patterns in your projects.

Next stop: Database operations, APIs, and workflow orchestration.