

A tool definition becomes a tested, read-only integration

You practise this in the course

server.py (excerpt)

```
from mcp.server import MCPServer

mcp = MCPServer("demo-orders")

@mcp.tool()
def find_orders(customer_id: str, since: str) -> list[dict]:
    """Find a customer's orders since a date (YYYY-MM-DD)."""
    check_inputs(customer_id, since) # rejects bad input
    return query_sample_orders(customer_id, since)

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Tests

Request	Expected	Result
C-1042 since 2026-09-01	Three orders, structured	Passed
since "last month"	Error: date must be YYYY-MM-DD	Passed
C-9999 (unknown)	Empty list	Passed
Log output	No personal data	Passed after fix

Security notes

- Read-only: no write or delete tools
- API key read from the environment, not the code
- Errors describe the input problem only

YOU START WITH

An integration design: one read-only tool over fictional order data.

THE PROMPT USED

“Help me add input validation to find_orders: customer_id must look like C-1234 and since must be a valid YYYY-MM-DD date. Return a clear error for bad input without exposing internal details, and suggest tests for valid, invalid and unknown-customer requests.”

STILL NEEDS HUMAN CHECKING

- That error messages leak no internal detail
- That logs contain no personal data
- That there are no write paths

WHAT YOU PRACTISE

Defining a tool with typed inputs, validating them, returning structured results and testing valid, invalid and unknown requests.

THE FINISHED OUTPUT CONTAINS

- A tool with typed, validated inputs
- Structured results and clear errors
- A test log for valid, invalid and unknown requests

RELATED COURSE

[Building MCP Integrations for Claude](#) · Module: Building with the official SDK · Security and reliability · Testing and deployment

Account note: Python 3.10 or later with uv and the official MCP Python SDK; a TypeScript version is also available. Code shown is an excerpt; helper functions are written in the course.