

Lab 016: GitHub Copilot Agent Mode

Level: L100 Path: ? GitHub Copilot Time: ~30 min ? Cost: GitHub Free ? Free GitHub account (free tier includes agent mode)

What You'll Learn

What makes agent mode different from regular Copilot Chat How to activate and use agent mode in VS Code How the agent reads your codebase, plans, and executes multi-step tasks How to connect MCP servers to expand agent capabilities Best practices and limitations

Introduction

GitHub Copilot in VS Code has three modes:

Mode | What it does | Ask | Answers questions about code; read-only | Plan | Explores the repository and proposes a reviewable implementation plan | Agent ? | Autonomously explores your codebase, runs commands, uses tools, and completes multi-step tasks |

Agent mode is the newest and most powerful. You describe a goal, and Copilot acts like a junior developer: it reads files, writes code, runs tests, and iterates until done ? asking for your approval at key decision points.

info "Available in VS Code 1.99+" Agent mode requires VS Code 1.99 or later and GitHub Copilot extension. Check for updates if you don't see the mode switcher.

Prerequisites Setup

1. VS Code 1.99+ with GitHub Copilot extension installed 2. Free GitHub account with Copilot enabled (github.com/features/copilot) 3. A project to work with (we'll use a simple Python project)

tip "Quick Start with GitHub Codespaces" Open in GitHub Codespaces

All dependencies are pre-installed in the devcontainer.

? Supporting Files

note "Download these files before starting the lab" Save all files to a lab-016/ folder in your working directory.

File | Description | Download | [outdoorgear_api.py](#) | Python script | ? Download |

Lab Exercise

Step 1: Activate agent mode

1. Open the Copilot Chat panel (Ctrl+Shift+I) 2. Look for the mode switcher at the top of the chat input 3. Select "Agent"

You'll notice the input changes ? you can now describe goals, not just ask questions.

Step 2: The Broken Project ? Fix it with Agent Mode ?

This exercise gives you a real broken Python project to fix using agent mode. The goal is to see how the agent reads files, identifies problems, and fixes them ? step by step.

Download the project: [\[diagram or code sample\]](#)

Or copy the file ? [outdoorgear_api.py](#) below:

[\[diagram or code sample\]](#)

Open the folder in VS Code (important ? the agent needs to see the whole project): [\[diagram or code sample\]](#)

Phase 1: Let the agent find and fix the bugs

Switch to Agent mode and type exactly this:

[\[diagram or code sample\]](#)

Watch what the agent does:

1. ? It reads the file without you pasting anything 2. ? It identifies each bug and explains why it's wrong 3. ?? It proposes fixes and asks your approval 4. ?? It runs the file to verify the fix worked

After accepting, run the verification: [\[diagram or code sample\]](#) Tests 1?6 should pass. Test

7 will fail (that's expected ? the function is missing).

tip "If the agent gets stuck" Try being more specific: "Run python outdoorgear_api.py and show me the error output, then fix the remaining bug"

Phase 2: Add the missing feature

Now ask the agent to implement the missing search_by_price_range function:

[diagram or code sample]

The agent should: 1. Read the existing code to understand the data structures 2. Implement the function 3. Run the tests to verify

Phase 3: Write a test suite

Now ask the agent to create proper tests:

[diagram or code sample]

Watch the agent: Creates the tests/ folder Writes comprehensive tests using pytest fixtures

Runs pytest in the terminal Fixes any test failures it finds

Phase 4: Improve code quality

[diagram or code sample]

Step 3: Codebase exploration

Try asking the agent to analyze what it just created:

[diagram or code sample]

The agent reads the whole codebase and synthesizes a coherent answer ? without you pasting any code.

Step 4: Connect an MCP server (bonus)

Agent mode supports MCP servers. Configure VS Code to use the MCP server from Lab 020:

.vscode/mcp.json: [diagram or code sample]

Then ask in agent mode: [diagram or code sample]

Step 5: Custom instructions

Create .github/copilot-instructions.md to make the agent always follow your project conventions:

[diagram or code sample]

Plan Mode vs. Agent Mode: When to Use Which

Use Plan mode when | Use Agent mode when | You need to understand scope before changing files | You have approved the approach and want it implemented | Requirements or affected files are unclear | The task requires multi-file, multi-step execution | You want a

reviewable plan and checkpoints | You want the agent to run tools and verify the result |

Architecture and trade-offs still need decisions | The expected behavior and boundaries are clear |

What the Agent Did (Behind the Scenes)

[diagram or code sample]

Summary

? Reads your codebase ? no copying/pasting code into chat ? Multi-step execution ? plans

and completes complex tasks ? Terminal access ? runs tests, verifies fixes ? MCP

integration ? connect custom tools ? Approvals at every step ? you stay in control

Next Steps

Build an MCP server to extend agent mode: ? Lab 020 ? MCP Server in Python Build a VS Code

Chat Participant (custom @agent): ? Lab 025 ? VS Code Chat Participant

Supporting kit

The site build generates stable PDF and ZIP downloads. The ZIP contains the lab guide and every supporting file declared for this lab.

Download generated PDF Download generated ZIP

Supporting file: outdoorgear_api.py

outdoorgear_api.py ? BROKEN PROJECT ? Use Copilot Agent Mode to fix and extend this SCENARIO: This is supposed to be the OutdoorGear Inc. product management API. It was written in a hurry and has several problems: - 5 bugs that prevent it from running -

Missing features that are referenced but not implemented - No tests - Poor error handling

YOUR MISSION (using Copilot Agent Mode): Phase 1: "Fix all the bugs in this file so the tests pass" Phase 2: "Add the missing search_by_price_range function" Phase 3: "Create a

tests/ folder with pytest tests for all functions" Phase 4: "Add type hints and docstrings to all public functions" HOW TO USE: 1. Open this FOLDER in VS Code (not just this file) 2.

Open Copilot Chat ? switch to AGENT MODE 3. Try Phase 1 first: "Fix all the bugs in

outdoorgear_api.py so the basic tests at the bottom pass when I run python outdoorgear_api.py" 4. Watch the agent: it will read the file, identify bugs, and fix them

5. Continue with Phase 2, 3, 4

```
from datetime import datetime import json
```

```
CATALOG = [ {"id": "BOOT-001", "name": "TrailBlazer X200", "category": "footwear", "price": 189.99, "stock": 42, "active": True}, {"id": "TENT-001", "name": "Summit Pro Tent", "category": "camping", "price": 349.00, "stock": 15, "active": True}, {"id": "HRNS-001", "name": "ClimbTech Pro Harness", "category": "climbing", "price": 129.99, "stock": 28, "active": True}, {"id": "PACK-001", "name": "OmniPack 45L", "category": "packs", "price": 279.99, "stock": 31, "active": True}, {"id": "JACK-001", "name": "StormShell Jacket", "category": "clothing", "price": 349.00, "stock": 8, "active": True}, {"id": "BOOT-002", "name": "Summit Trail Low", "category": "footwear", "price": 99.99, "stock": 55, "active": False}, ]
```

```
ORDERS = []
```

```
def get_all_products(include_inactive: bool = False) -> list: """Return all products, optionally including inactive ones.""" if include_inactive: return CATALOG return [p for p in CATALOG if p["active"] == True]
```

```
def get_product_by_id(product_id: str) -> dict | None: """Return a product by its ID, or None if not found.""" for product in CATALOG: if product["id"] = product_id: # BUG 1: assignment instead of comparison return product return None
```

```
def add_to_cart(cart: dict, product_id: str, quantity: int) -> dict: """Add a product to the cart. Returns updated cart.""" product = get_product_by_id(product_id) if product is None: raise ValueError(f"Product {product_id} not found") if product["stock"] float: """Calculate total price for all items in cart.""" total = 0 for item in cart: # BUG 3: iterating over keys, not values total += item["product"]["price"] * item["quantity"] return round(total, 2)
```

```
def apply_promo_code(total: float, code: str) -> float: """Apply a promo code discount to the total.""" VALID_CODES = { "SUMMER10": 0.10, "OUTDOOR20": 0.20, "WELCOME5": 0.05, } if code in VALID_CODES: discount = total * VALID_CODES[code] return round(total - discount, 2) else: raise ValueError(f"Invalid promo code: {code}") # BUG 4: should return total unchanged, not raise
```

```
def place_order(cart: dict, customer_name: str, promo_code: str = None) -> dict: """Convert cart to an order. Returns order confirmation.""" if len(cart) == 0: # BUG 5: should use "not cart" or "len(cart) == 0" raise ValueError("Cannot place empty order") # actually this is fine, but...
```

```
total = calculate_cart_total(cart)
```

```
if promo_code: total = apply_promo_code(total, promo_code)
```

```

order = { "order_id": f"ORD-{len(ORDERS) + 1:04d}", "customer": customer_name, "items":
cart, "total": total, "placed_at": datetime.now().isoformat(), "status": "confirmed", }
# Deduct stock for product_id, item in cart.item(): # BUG 5: .item() should be .items() for
product in CATALOG: if product["id"] == product_id: product["stock"] -= item["quantity"]
ORDERS.append(order) return order
--- MISSING FEATURE --- The search_by_price_range function is referenced in the tests below
but hasn't been implemented yet. Parameters: min_price: float, max_price: float Returns:
list of products in that price range (inclusive), sorted by price ascending
===== Basic tests ? these should
pass after Phase 1 ===== if __name__
== "__main__": print("Running basic tests...\n")
# Test 1: get all active products active = get_all_products() assert len(active) == 5,
f"Expected 5 active products, got {len(active)}" print(f"? Test 1: get_all_products() ? 5
active products")
# Test 2: get product by ID boot = get_product_by_id("BOOT-001") assert boot is not None,
"BOOT-001 should exist" assert boot["name"] == "TrailBlazer X200" print(f"? Test 2:
get_product_by_id() ? found BOOT-001")
# Test 3: add to cart and calculate total cart = {} cart = add_to_cart(cart, "BOOT-001", 2)
cart = add_to_cart(cart, "TENT-001", 1) total = calculate_cart_total(cart) assert abs(total
- 728.98) < 0.01, f"Expected $728.98, got ${total}" print(f"? Test 3: cart total =
${total}")
# Test 4: promo code discounted = apply_promo_code(728.98, "SUMMER10") assert
abs(discounted - 656.08) < 0.01, f"Expected $656.08, got ${discounted}" print(f"? Test 4:
promo SUMMER10 applied = ${discounted}")
# Test 5: invalid promo code should NOT raise ? should return original total unchanged =
apply_promo_code(728.98, "INVALID_CODE") assert unchanged == 728.98, f"Invalid code should
return original total, got ${unchanged}" print(f"? Test 5: invalid promo code returns
original total = ${unchanged}")
# Test 6: place order order = place_order(cart, "Alex Chen", "SUMMER10") assert
order["status"] == "confirmed" assert order["customer"] == "Alex Chen" print(f"? Test 6:
order placed ? {order['order_id']}")
# Test 7: search by price range (requires Phase 2) budget_items = search_by_price_range(50,
200) assert len(budget_items) == 2 # TrailBlazer X200 ($189.99) and ClimbTech ($129.99)
assert budget_items[0]["price"] <= budget_items[1]["price"] # sorted ascending print(f"?
Test 7: search_by_price_range(50, 200) ? found {len(budget_items)} products")
print("\n? All tests passed! Ready for Phase 3 (write tests) and Phase 4 (type hints).")

```