

Lab 016: GitHub Copilot Agent Mode

Nível: L100 Trilha: ? GitHub Copilot Tempo: ~30 min ? Custo: GitHub Free ? Conta GitHub gratuita (o plano gratuito inclui o agent mode)

O que você vai aprender

O que torna o agent mode diferente do Copilot Chat convencional Como ativar e usar o agent mode no VS Code Como o agente lê sua base de código, planeja e executa tarefas em múltiplas etapas Como conectar servidores MCP para expandir as capacidades do agente Boas práticas e limitações

Introdução

O GitHub Copilot no VS Code possui três modos:

Modo | O que faz | Ask | Responde perguntas sobre código; somente leitura | Plan | Explora o repositório e propõe um plano de implementação revisável | Agent ? | Explora autonomamente sua base de código, executa comandos, usa ferramentas e conclui tarefas em múltiplas etapas |

O Agent mode é o mais novo e mais poderoso. Você descreve um objetivo, e o Copilot age como um desenvolvedor júnior: ele lê arquivos, escreve código, executa testes e itera até concluir ? pedindo sua aprovação em pontos-chave de decisão.

info "Disponível no VS Code 1.99+" O Agent mode requer VS Code 1.99 ou posterior e a extensão GitHub Copilot. Verifique se há atualizações caso você não veja o seletor de modo.

Configuração de Pré-requisitos

1. VS Code 1.99+ com a extensão GitHub Copilot instalada 2. Conta GitHub gratuita com Copilot habilitado (github.com/features/copilot) 3. Um projeto para trabalhar (usaremos um projeto Python simples)

tip "Início Rápido com GitHub Codespaces" Open in GitHub Codespaces

Todas as dependências já estão pré-instaladas no devcontainer.

? Arquivos de Apoio

note "Baixe estes arquivos antes de iniciar o laboratório" Salve todos os arquivos em uma pasta lab-016/ no seu diretório de trabalho.

Arquivo | Descrição | Download | [outdoorgear_api.py](#) | Script Python | ? Download |

Exercício do Laboratório

Etapa 1: Ativar o agent mode

1. Abra o painel do Copilot Chat (Ctrl+Shift+I) 2. Procure o seletor de modo na parte superior da entrada do chat 3. Selecione "Agent"

Você vai notar que a entrada muda ? agora é possível descrever objetivos, não apenas fazer perguntas.

Etapa 2: O Projeto Quebrado ? Corrija com Agent Mode ?

Este exercício oferece um projeto Python realmente quebrado para corrigir usando o agent mode. O objetivo é ver como o agente lê arquivos, identifica problemas e os corrige ? passo a passo.

Baixe o projeto: [diagram or code sample]

Ou copie o arquivo ? [outdoorgear_api.py](#) abaixo:

[diagram or code sample]

Abra a pasta no VS Code (importante ? o agente precisa ver o projeto inteiro): [diagram or code sample]

Fase 1: Deixe o agente encontrar e corrigir os bugs

Mude para o Agent mode e digite exatamente isto:

[diagram or code sample]

Observe o que o agente faz:

1. ? Ele lê o arquivo sem que você precise colar nada 2. ? Ele identifica cada bug e explica por que está errado 3. ?? Ele propõe correções e pede sua aprovação 4. ?? Ele executa o arquivo para verificar se a correção funcionou

Após aceitar, execute a verificação: [diagram or code sample] Os testes 1?6 devem passar. O teste 7 vai falhar (isso é esperado ? a função está ausente).

tip "Se o agente travar" Tente ser mais específico: "Run python outdoorgear_api.py and show me the error output, then fix the remaining bug"

Fase 2: Adicionar a funcionalidade ausente

Agora peça ao agente para implementar a função search_by_price_range que está faltando:

[diagram or code sample]

O agente deve: 1. Ler o código existente para entender as estruturas de dados 2.

Implementar a função 3. Executar os testes para verificar

Fase 3: Escrever um conjunto de testes

Agora peça ao agente para criar testes adequados:

[diagram or code sample]

Observe o agente: Criar a pasta tests/ Escrever testes abrangentes usando fixtures do

pytest Executar pytest no terminal Corrigir quaisquer falhas de teste encontradas

Fase 4: Melhorar a qualidade do código

[diagram or code sample]

Etapa 3: Exploração da base de código

Tente pedir ao agente para analisar o que ele acabou de criar:

[diagram or code sample]

O agente lê toda a base de código e sintetiza uma resposta coerente ? sem que você precise colar nenhum código.

Etapa 4: Conectar um servidor MCP (bônus)

O Agent mode suporta servidores MCP. Configure o VS Code para usar o servidor MCP do Lab 020:

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

Em seguida, pergunte no agent mode: [diagram or code sample]

Etapa 5: Instruções personalizadas

Crie .github/copilot-instructions.md para fazer o agente sempre seguir as convenções do seu projeto:

[diagram or code sample]

Modo Plan vs. Modo Agent: Quando Usar Cada Um

Use o modo Plan quando | Use o modo Agent quando | Você precisa entender o escopo antes de alterar arquivos | Você aprovou a abordagem e quer implementá-la | Requisitos ou arquivos afetados não estão claros | A tarefa exige vários arquivos e etapas | Você quer um plano revisável e checkpoints | Você quer que o agente execute ferramentas e valide o resultado | Arquitetura e trade-offs ainda exigem decisões | O comportamento esperado e os limites estão claros |

O que o Agente Fez (Por Trás dos Bastidores)

[diagram or code sample]

Resumo

? Lê sua base de código ? sem copiar/colar código no chat ? Execução em múltiplas etapas ? planeja e conclui tarefas complexas ? Acesso ao terminal ? executa testes, verifica correções ? Integração com MCP ? conecte ferramentas personalizadas ? Aprovações a cada etapa ? você mantém o controle

Próximos Passos

Construa um servidor MCP para estender o agent mode: ? Lab 020 ? MCP Server in Python
Construa um Chat Participant do VS Code (@agent personalizado): ? Lab 025 ? VS Code Chat Participant
Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado

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).")
```