

warning "Mise à jour du backend (octobre 2026)" L'inférence GitHub Models a été retirée le 30 juillet 2026. Les références ci-dessous sont conservées uniquement comme contexte historique pendant la migration de ce lab. Utilisez le Lab 013 ? Backends LLM gratuits pour configurer Foundry Local ou Ollama avant de continuer.

Lab 010 : GitHub Copilot ? Premiers pas

Niveau : L100 Parcours : ? GitHub Copilot Durée : ~45 min ? Coût : GitHub Free ? quota de complétions et allocation mensuelle limitée d'AI Credits

Ce que vous apprendrez

Utiliser la complétion de code en ligne pour écrire du code à partir de commentaires

Utiliser Copilot Chat /fix pour trouver et comprendre de vrais bugs Utiliser le mode Agent

pour refactoriser un fichier entier en langage naturel Utiliser le chat en ligne pour

étendre le code sans quitter l'éditeur Écrire des prompts qui donnent de meilleurs résultats

Ce lab utilise des exercices pratiques ? vous ouvrirez des fichiers contenant de vrais bugs

et du code incomplet, puis vous utiliserez Copilot pour les corriger et les étendre.

Prérequis

1. Activer GitHub Copilot Free

1. Rendez-vous sur github.com/features/copilot ? « Commencer gratuitement » 2.

Connectez-vous et suivez l'assistant de configuration

info "La disponibilité pour les étudiants évolue" Copilot Free est le parcours fiable pour ce lab. Consultez le GitHub Student Developer Pack pour vérifier l'éligibilité actuelle, sans supposer qu'un abonnement payant est inclus.

2. Installer VS Code + l'extension Copilot

1. Installez VS Code 2. Extensions (Ctrl+Shift+X) ? recherchez « GitHub Copilot » ?

Installez les deux : - GitHub Copilot (complétions) - GitHub Copilot Chat (panneau de chat)

3. Connectez-vous lorsque vous y êtes invité ? vous verrez l'icône Copilot dans la barre d'état

3. Télécharger les fichiers d'exercices

Clonez ou téléchargez les fichiers d'exercices pour ce lab :

[diagram or code sample]

Ou copiez chaque fichier directement depuis les sections ci-dessous.

tip "Démarrage rapide avec GitHub Codespaces" Open in GitHub Codespaces

Toutes les dépendances sont pré-installées dans le devcontainer.

? Fichiers de support

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-010/ de votre répertoire de travail.

Fichier | Description | Téléchargement | [exercice1_fibonacci.py](#) | Script d'exercice

interactif | ? Télécharger | [exercice2_shopping_cart.py](#) | Script d'exercice interactif | ?

Télécharger | [exercice3_product_search.py](#) | Script d'exercice interactif | ? Télécharger |

[exercice4_refactor_me.py](#) | Script d'exercice interactif | ? Télécharger |

Exercice 1 ? Complétion en ligne : Écrire du code à partir de commentaires

Objectif : Apprendre comment Copilot complète le code au fur et à mesure que vous tapez.

Créez un nouveau fichier `practice.py` et tapez chaque commentaire ci-dessous. Après chaque commentaire, arrêtez de taper et attendez la suggestion de Copilot. Appuyez sur Tab pour accepter, vous pouvez continuer à appuyer sur Tab jusqu'à ce que Copilot arrête de proposer des complétions.

tip "Nous n'exécuterons pas ce code, donc ne vous inquiétez pas des erreurs de syntaxe ou des imports manquants ? concentrez-vous uniquement sur les suggestions que Copilot vous donne en fonction des commentaires."

[diagram or code sample]

tip "Raccourcis clavier" | Touche | Action | |-----|-----| | Tab | Accepter la suggestion | | Esc | Ignorer | | Alt+ | / Alt+ | Suggestion suivante / précédente | | Ctrl+Enter | Ouvrir le panneau de toutes les suggestions |

Essayez des prompts meilleurs et moins bons :

? Vague | ? Spécifique | # sort this | # Sort list of dicts by 'price' descending, then 'name' ascending | # connect to db | # Connect to PostgreSQL using asyncpg, return a connection pool | # handle error | # Retry 3 times with exponential backoff if requests.Timeout is raised |

Exercice 2 ? Copilot /fix : Chasse aux bugs ?

Objectif : Utiliser Copilot Chat pour trouver, comprendre et corriger de vrais bugs.

Fichier : [? exercise1_fibonacci.py

[diagram or code sample]

Étapes :

1. Copiez le code ci-dessus dans un nouveau fichier (ou ouvrez-le à partir des exercices téléchargés) 2. Ouvrez Copilot Chat (Ctrl+Shift+I) 3. Sélectionnez tout le code (Ctrl+A) 4.

Tapez : /fix

Copilot devrait identifier les 3 bugs et expliquer chacun d'entre eux. Avant d'accepter, lisez l'explication ? comprendre *pourquoi* le code était incorrect, c'est tout l'intérêt.

Sortie attendue après correction : [diagram or code sample]

Exécutez python exercise1_fibonacci.py ? vous devriez voir : ? All tests passed!

Fichier : ? exercise2_shopping_cart.py

[diagram or code sample]

Ce fichier contient 4 bugs dans la classe ShoppingCart. Cette fois, avant d'utiliser /fix :

1. Essayez d'abord de repérer les bugs vous-même ? passez 2 minutes à lire le code 2. Puis utilisez Copilot Chat : sélectionnez tout ? /fix 3. Copilot a-t-il trouvé des bugs que vous avez manqués ?

Demandez à Copilot d'expliquer un bug en profondeur : [diagram or code sample]

Sortie attendue après correction : [diagram or code sample]

Exercice 3 ? Chat en ligne : Corriger + Étendre

Objectif : Corriger des bugs ET ajouter une nouvelle fonctionnalité en utilisant le chat en ligne (Ctrl+I).

Fichier : ? exercise3_product_search.py

[diagram or code sample]

Partie A ? Correction (2 bugs) :

1. Ouvrez le fichier dans VS Code 2. Sélectionnez tout (Ctrl+A) ? Copilot Chat ? /fix 3.

Vérifiez : python exercise3_product_search.py ? les tests 1 à 4 devraient passer

Partie B ? Extension (1 fonctionnalité manquante) :

Le fichier mentionne une fonction sort_by_price() qui n'existe pas encore.

1. Placez votre curseur à la fin du fichier (avant la section des tests) 2. Appuyez sur Ctrl+I (chat en ligne) 3. Tapez exactement : [diagram or code sample] 4. Examinez la suggestion et appuyez sur Accepter (Ctrl+Enter) 5. Relancez les tests ? les 5 devraient maintenant passer

Exercice 4 ? Mode Agent : Refactoriser un fichier entier

Objectif : Utiliser le mode Agent pour améliorer la qualité du code avec des instructions en langage naturel ? sans changer le comportement.

Fichier : ? exercise4_refactor_me.py

[diagram or code sample]

Ce code fonctionne correctement mais est difficile à lire et à maintenir. Utilisez le mode Agent pour l'améliorer étape par étape :

1. Ouvrez Copilot Chat (Ctrl+Shift+I) et sélectionnez Agent 2. Ajoutez `exercice4_refactor_me.py` au contexte du chat 3. Exécutez chacun de ces prompts un à la fois, en examinant les modifications avant de continuer :

Prompt 1 : [diagram or code sample]

Prompt 2 : [diagram or code sample]

Prompt 3 : [diagram or code sample]

Prompt 4 : [diagram or code sample]

Après chaque prompt, vérifiez que le test en bas du fichier passe toujours : [diagram or code sample]

warning "N'acceptez pas tout aveuglément" Parfois Copilot ajoute de la complexité superflue. Si une suggestion rend le code plus difficile à lire, appuyez sur Annuler (Ctrl+Backspace) et reformulez.

Bonus : Demandez à Copilot d'expliquer, pas seulement de corriger

Utilisez ces prompts sur n'importe quel fichier d'exercice pour approfondir votre compréhension :

[diagram or code sample] [diagram or code sample] [diagram or code sample] [diagram or code sample]

Ce que vous avez pratiqué

Fonctionnalité Copilot | Exercice | Cas d'utilisation | Complétion en ligne | Exercice 1 | Écrire du nouveau code à partir de commentaires | Chat /fix | Exercice 2 | Comprendre et corriger des bugs | Chat en ligne Ctrl+I | Exercice 3 | Corriger + étendre sur place | Mode Agent | Exercice 4 | Refactoriser des fichiers entiers et vérifier les modifications | Prochaines étapes

Construire un agent sans code pour Teams : ? Lab 011 ? Copilot Studio Utiliser le mode Agent pour construire une fonctionnalité complète : ? Lab 016 ? Copilot Agent Mode Utiliser des LLM gratuits dans votre code : ? Lab 013 ? GitHub Models

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré

Supporting file: `exercice1_fibonacci.py`

`exercice1_fibonacci.py` ? BUG HUNT ? This file has 3 bugs. Can you find them all using Copilot /fix? TASK: The `fibonacci()` function should return a list of the first `n` Fibonacci numbers: `[0, 1, 1, 2, 3, 5, 8, 13, ...]` EXPECTED OUTPUT (for `n=8`): `fibonacci(8)` ? `[0, 1, 1, 2, 3, 5, 8, 13]` `fibonacci(1)` ? `[0]` `fibonacci(0)` ? `[]` HOW TO USE THIS EXERCISE: 1. Open this file in VS Code 2. Select ALL the code (Ctrl+A) 3. Open Copilot Chat ? type: /fix 4. Read Copilot's explanation of each bug 5. Accept the fix and run the tests below to verify

```
def fibonacci(n): """Return a list of the first n Fibonacci numbers.""" if n = 0: # BUG 1:
syntax error return []
sequence = [0, 1]
for i in range(2, n): next_val = sequence[i-1] + sequence[i-2] sequence.append(next_val)
return sequence[n] # BUG 2: should return a slice, not a single element # BUG 3: what
happens when n=1? sequence has 2 elements but we only want 1
--- Tests (run after fixing) --- if __name__ == "__main__": assert fibonacci(0) == [],
f"Expected [], got {fibonacci(0)}" assert fibonacci(1) == [0], f"Expected [0], got
```

```
{fibonacci(1)}" assert fibonacci(8) == [0, 1, 1, 2, 3, 5, 8, 13], f"Got {fibonacci(8)}"
print("? All tests passed!")
```

Supporting file: exercise2_shopping_cart.py

exercise2_shopping_cart.py ? BUG HUNT ? This file has 4 bugs. Use Copilot Chat to find and fix them. SCENARIO: OutdoorGear Inc. shopping cart logic. The cart should: - Add items with a quantity - Calculate total price (sum of price × quantity) - Apply a 10% discount on orders over \$200 - Count unique items (not total quantity) EXPECTED OUTPUT: Cart with 2x TrailBlazer (\$189.99) + 1x Summit Tent (\$349): total before discount = \$728.98 total after 10% discount = \$656.08 unique_items = 2 HOW TO USE: 1. Copy this file into VS Code 2. Try to run it: python exercise2_shopping_cart.py 3. Read the errors ? but DON'T fix manually yet! 4. Select all code ? Copilot Chat ? /fix 5. Ask Copilot to explain EACH bug before fixing 6. Verify with the tests at the bottom

```
class ShoppingCart:
    def __init__(self):
        self.items = {} # {product_name: {"price": float, "quantity": int}}

    def add_item(self, name: str, price: float, quantity: int = 1):
        if name in self.items:
            self.items[name]["quantity"] += quantity
        else:
            self.items[name] = {"price": price, "quantity": quantity}

    def get_total(self) -> float:
        """Calculate total price, applying 10% discount if over $200.
        total = 0 for item in self.items: # BUG 1: iterating over keys only, not values
            total += item["price"] * item["quantity"]
        if total > 200:
            total = total * 0.90 # BUG 2: discount calculation correct, but... # should
            return rounded to 2 decimal places
        return total

    def count_unique_items(self) -> int:
        """Return the number of unique products (not total quantity).
        return sum(self.items.values()) # BUG 3: wrong ? should be len(), not sum()

    def get_receipt(self) -> str:
        lines = []
        for name, details in self.items.items():
            subtotal = details["price"] * details["quantity"] # BUG 4: + instead of *
            lines.append(f" {name} x{details['quantity']} @ ${details['price']:.2f} = ${subtotal:.2f}")
        total = self.get_total()
        lines.append(f"\nTotal: ${total:.2f}")
        return "\n".join(lines)

--- Tests ---
if __name__ == "__main__":
    cart = ShoppingCart()
    cart.add_item("TrailBlazer X200", 189.99, 2)
    cart.add_item("Summit Pro Tent", 349.00, 1)
    total = cart.get_total()
    unique = cart.count_unique_items()
    print(cart.get_receipt())
    print(f"\nUnique items: {unique}")
    assert unique == 2, f"Expected 2 unique items, got {unique}"
    assert abs(total - 656.08) < 0.01, f"Expected $656.08, got ${total}"
    print("\n? All tests passed!")
```

Supporting file: exercise3_product_search.py

exercise3_product_search.py ? BUG HUNT + EXTEND ? This file has 2 bugs AND is missing a feature. SCENARIO: OutdoorGear product search with filtering. TASK A (Fix): Find and fix the 2 bugs using Copilot /fix TASK B (Extend): Ask Copilot to add a sort_by_price() function Prompt to use: "Add a sort_by_price(products, ascending=True) function that sorts the search results by price" EXPECTED OUTPUT after fixing: search("boot") ? [{"name": "TrailBlazer X200", "price": 189.99, "category": "footwear"}] search("tent") ? [{"name": "Summit Pro Tent", "price": 349.00, "category": "camping"}] search("xyz") ? [] search("") ? all 5 products HOW TO USE: Step 1 ? Fix bugs: Select all ? Copilot Chat ? /fix Step 2 ? Extend: Ctrl+I (inline chat) ? type the prompt above Step 3 ? Test: Run the tests at the bottom

```
CATALOG = [{"name": "TrailBlazer X200", "price": 189.99, "category": "footwear"}, {"name":
```

```

"Summit Pro Tent", "price": 349.00, "category": "camping"}, {"name": "ClimbTech Pro
Harness", "price": 129.99, "category": "climbing"}, {"name": "OmniPack 45L", "price":
279.99, "category": "packs"}, {"name": "StormShell Jacket", "price": 349.00, "category":
"clothing"}, ]
def search_products(query: str) -> list[dict]: """Return products matching query in name or
category (case-insensitive).""" if not query: return CATALOG
query_lower = query.lower() results = [] for product in CATALOG: if query_lower in
product["name"].lower or \ # BUG 1: missing () on .lower query_lower in
product["category"].lower(): result.append(product) # BUG 2: wrong variable name (result vs
results) return results
def format_results(products: list[dict]) -> str: """Format search results for display."""
if not products: return "No products found." lines = [f"Found {len(products)} product(s):"]
for p in products: lines.append(f" ? {p['name']} ? ${p['price']:.2f} ({p['category']})")
return "\n".join(lines)
--- Tests --- if __name__ == "__main__": print("Test 1: search for 'boot'") r1 =
search_products("boot") print(format_results(r1)) assert len(r1) == 1 and r1[0]["name"] ==
"TrailBlazer X200"
print("\nTest 2: search for 'tent'") r2 = search_products("tent") print(format_results(r2))
assert len(r2) == 1 and r2[0]["name"] == "Summit Pro Tent"
print("\nTest 3: empty search returns all") r3 = search_products("") assert len(r3) == 5
print("\nTest 4: search 'xyz' returns empty") r4 = search_products("xyz") assert r4 == []
# Test the sort function Copilot should have added: print("\nTest 5: sort by price
ascending") sorted_asc = sort_by_price(CATALOG, ascending=True) assert
sorted_asc[0]["name"] == "ClimbTech Pro Harness" # cheapest first assert
sorted_asc[-1]["price"] == 349.00
print("\n? All tests passed!")

```

Supporting file: exercise4_refactor_me.py

exercise4_refactor_me.py ?? REFACTORING CHALLENGE ? No bugs here, but the code is terrible.

TASK: Use Copilot Edits to improve this code WITHOUT changing its behavior. TRY THESE
 COPILOT EDITS PROMPTS (one at a time): 1. "Add type hints to all function parameters and
 return values" 2. "Add docstrings following Google style to every function" 3. "Replace the
 magic numbers with named constants at the top of the file" 4. "Refactor calculate_shipping
 to use early return instead of nested ifs" 5. "Add input validation: raise ValueError for
 negative prices or quantities" After each prompt, review what Copilot changed. Accept or
 reject with Ctrl+Enter / Ctrl+Backspace. GOAL: By the end, this code should be
 production-ready.

```

STANDARD_SHIPPING = 5.99 EXPRESS_SHIPPING = 14.99 FREE_SHIPPING_THRESHOLD = 75.0
DISCOUNT_THRESHOLD = 200.0 DISCOUNT_RATE = 0.10

```

```

def calculate_price(p, q): return p * q
def calculate_shipping(t): if t >= 75: return 0 else: if t > 50: return 5.99 else: return
14.99
def apply_discount(t): if t > 200: d = t * 0.10 t = t - d return round(t, 2)
def order_summary(items): t = 0 for i in items: t = t + calculate_price(i[0], i[1]) s =
calculate_shipping(t) t_with_shipping = t + s final = apply_discount(t_with_shipping)
return {"subtotal": round(t, 2), "shipping": s, "discount_applied": t > 200, "total": final}
Test ? this should still pass after all refactoring if __name__ == "__main__": items =
[(189.99, 1), (349.00, 1)] # (price, quantity) result = order_summary(items) print(result)

```

```
assert result["total"] == 489.59, f'Got {result["total"]}' print(f'? Refactoring complete ?  
behavior unchanged!')
```