

Lab 023: Outils et mémoire dans Microsoft Agent Framework

Level: L200 Parcours: ai-agents Durée: ~90 min Coût: free

info "Phase 3 · local first" Le parcours obligatoire est déterministe et local. Le parcours

Réel est facultatif, estimé et nettoyé.

Ce que vous apprendrez

Exposer des outils typés avec autorisation explicite Utiliser une mémoire courte et durable bornée Remplacer les planners SK obsolètes par une orchestration explicite Tester injection et isolation de mémoire

Prérequis

Python 3.10+ et un terminal Aucun identifiant pour le parcours obligatoire Lisez la spécification officielle avant un adaptateur hébergé

Modèle visuel

[diagram or code sample]

Scénario

OutdoorGear a besoin d'une implémentation révisable pour atelier, avec fixtures déterministes, frontières de confiance explicites, critères mesurables et adaptateur hébergé facultatif et réversible.

Exercice guidé

Step 1: define_tools

1. Exécutez la fixture déterministe de cette étape. 2. Notez entrée, sortie, décision de politique et comportement d'échec. 3. Modifiez un comportement borné et relancez le test local.

Evidence: Conservez la commande, le résultat et une phrase expliquant pourquoi la preuve est fiable.

Step 2: bound_memory

1. Exécutez la fixture déterministe de cette étape. 2. Notez entrée, sortie, décision de politique et comportement d'échec. 3. Modifiez un comportement borné et relancez le test local.

Evidence: Conservez la commande, le résultat et une phrase expliquant pourquoi la preuve est fiable.

Step 3: route_explicitly

1. Exécutez la fixture déterministe de cette étape. 2. Notez entrée, sortie, décision de politique et comportement d'échec. 3. Modifiez un comportement borné et relancez le test local.

Evidence: Conservez la commande, le résultat et une phrase expliquant pourquoi la preuve est fiable.

Step 4: red_team

1. Exécutez la fixture déterministe de cette étape. 2. Notez entrée, sortie, décision de politique et comportement d'échec. 3. Modifiez un comportement borné et relancez le test local.

Evidence: Conservez la commande, le résultat et une phrase expliquant pourquoi la preuve est fiable.

Exécuter localement

[diagram or code sample]

Simulation / Parcours Réel

Simulation Parcours Réel

Utilisez les fixtures déterministes et le validateur local. Coût estimé : 0 USD.

Remplacez uniquement la frontière de l'adaptateur. Fixez un plafond, gardez les mêmes tests

et notez région, rétention, identité, télémétrie et prix. Supprimez déploiements, points de terminaison, index, traces et données de test après validation.

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. La CI conserve aussi un miroir temporaire pendant 30 jours.

README.md : démarrage et preuves CHEATSHEET.md : guide de décision scenario.json : contrat déterministe test_scenario.py : tests locaux Télécharger le PDF généré Télécharger le ZIP généré

Quiz noté

question "Pourquoi garder un contrat local ?"

??? success "Réponse" Il fournit une base déterministe avant la variabilité et le coût hébergés.

question "Quand le parcours Réel est-il terminé ?"

??? success "Réponse" Après enregistrement des preuves de validation et de nettoyage.

question "Comment traiter la sortie des outils ?"

??? success "Réponse" Comme donnée non fiable avant validation de politique et de schéma.

Critères de sortie

[] Les tests locaux passent sans identifiants. [] Le diagramme a une explication textuelle et ne dépend pas de la couleur. [] Le quiz noté a été revu. [] Le parcours Réel a un plafond et un plan de nettoyage. [] Sources et stabilité ont été vérifiées.

Sources publiques officielles

Microsoft Agent Framework 1.0 GA AutoGen to Agent Framework migration

Supporting file: CHEATSHEET.md

Outils et mémoire dans Microsoft Agent Framework ? Cheat Sheet

Commencez par le contrat local et des critères explicites. Traitez prompts, contenu récupéré, outils et messages comme non fiables. Rendez identité, autorisation, politique, télémétrie, coût et nettoyage explicites. Préférez l'orchestration ou l'adaptation minimale suffisante.

Supporting file: README.md

Outils et mémoire dans Microsoft Agent Framework

python -m unittest discover -v

Le parcours obligatoire est local et déterministe.

Supporting file: broken_plugin.py

""" broken_plugin.py ? OutdoorGear SK Plugin ? INTENTIONALLY BROKEN

Lab 023: SK Plugins & Memory ? Interactive Bug-Fix Exercise

This file contains 3 deliberate bugs in the Semantic Kernel plugin implementation. Your task is to: 1. Read the BUG comment near each issue 2. Understand why it fails 3. Fix it ? then run the test at the bottom to verify

Prerequisites: pip install semantic-kernel openai export GITHUB_TOKEN=

Run: python broken_plugin.py

Expected output when all bugs are fixed: ? search_products: found 3 tents ? get_cart_total: \$539.98 ? calculate_price_with_tax: \$269.99 """

import asyncio import os

import semantic_kernel as sk from semantic_kernel.connectors.ai.openai import

OpenAIChatCompletion from semantic_kernel.functions import kernel_function

```

##### Data:
OutdoorGear product catalog ??????????????????????????????????????????????????????????????????
????????????????? PRODUCTS = [ {"id": "P001", "name": "TrailBlazer Tent 2P", "category":
"Tents", "price": 249.99}, {"id": "P002", "name": "Summit Dome 4P", "category": "Tents",
"price": 549.99}, {"id": "P003", "name": "TrailBlazer Solo", "category": "Tents", "price":
299.99}, {"id": "P004", "name": "ArcticDown -20°C Sleeping Bag", "category": "Sleeping
Bags", "price": 389.99}, {"id": "P005", "name": "SummerLight +5°C Sleeping Bag",
"category": "Sleeping Bags", "price": 149.99}, {"id": "P006", "name": "Osprey Atmos 65L
Backpack", "category": "Backpacks", "price": 289.99}, {"id": "P007", "name": "DayHiker 22L
Daypack", "category": "Backpacks", "price": 89.99}, ]
Shopping cart: list of (product_id, quantity) CART = [ ("P001", 1), # TrailBlazer Tent 2P ×
1 = $249.99 ("P007", 1), # DayHiker 22L Daypack × 1 = $89.99 # Expected total: $339.98 ]
???????????????????????????????????????????????????????????????????????????????????? The Plugin
class ????????????????????????????????????????????????????????????????????????????????? class
OutdoorGearPlugin: """ SK Plugin for querying the OutdoorGear product catalog and shopping
cart. """
# ===== # FUNCTION 1:
search_products ## ? BUG #1 ? Wrong decorator: missing @kernel_function ## Expected:
Semantic Kernel discovers and can invoke this function # Actual: SK ignores this method ?
it is not registered as a kernel function ## Hint: SK discovers plugin functions by
looking for the @kernel_function decorator. # Without it, the function is invisible to the
kernel. # Fix: Add @kernel_function(description= "...") above the def statement. #
===== # BUG #1:
@kernel_function decorator is missing ? add it here def search_products( self, keyword:
str, category: str = "", ) -> str: """Search for OutdoorGear products by keyword and
optional category.""" keyword_lower = keyword.lower() results = [ p for p in PRODUCTS if
keyword_lower in p["name"].lower() and (not category or p["category"] == category) ] if not
results: return f"No products found for keyword '{keyword}'\n\n" lines = [f"[{p['id']}]\n{p['name']}\n${p['price']: .2f}" for p in results] return "\n".join(lines)
# ===== # FUNCTION 2:
get_cart_total ## ? BUG #2 ? Logic error: calculates count instead of price total ##
Expected: Returns the sum of (price × quantity) for all cart items ? "$339.98" # Actual:
Returns the number of cart items ? "2" (always a small integer) ## Hint: The loop
accumulates quantity instead of price * quantity. # Fix: Change total += quantity to total
+= price * quantity. # =====
==== @kernel_function(description="Get the total price of all items in the shopping cart")
def get_cart_total(self) -> str: """Returns the current shopping cart total as a formatted
string.""" total = 0.0 for product_id, quantity in CART: product = next((p for p in
PRODUCTS if p["id"] == product_id), None) if product: price = product["price"] total +=
quantity ## ? BUG #2: should be total += price * quantity return f"${total:.2f}"
# ===== # FUNCTION 3:
calculate_price_with_tax ## ? BUG #3 ? Tax calculation applied twice (compounding error) #
# Expected: calculate_price_with_tax("P001", 0.08) # ? "$269.99" (249.99 × 1.08 = 269.9892
? $269.99) # Actual: ? "$291.59" (249.99 × 1.08 × 1.08 ? tax applied twice!) ## Hint: The
tax is added to the price once, then the result is multiplied # by (1 + tax_rate) again. #
Fix: Keep only one tax multiplication. #
=====
@kernel_function(description="Calculate the final price of a product including tax") def

```

```
calculate_price_with_tax( self, product_id: str, tax_rate: float = 0.08, ) -> str: """"
Calculate a product's final price after applying sales tax.
Args: product_id: The product ID (e.g. 'P001') tax_rate: Tax rate as a decimal (e.g. 0.08
for 8%). Default: 0.08 """" product = next((p for p in PRODUCTS if p["id"] == product_id),
None) if product is None: return f"Product '{product_id}' not found"
base_price = product["price"]
# BUG #3: tax is applied twice price_with_tax = base_price + (base_price * tax_rate) #
first tax application final_price = price_with_tax * (1 + tax_rate) # ? BUG: second tax
application
return f"${final_price:.2f}"
???????????????????????????????????????????????????????????????????????????????? Test runner
(verifies each function independently, no LLM needed for unit tests)
???????????????????????????????????????????????????????????????????????????????? def
run_unit_tests(): """Run tests directly on the plugin (no SK kernel needed).""" plugin =
OutdoorGearPlugin() all_passed = True
# Test 1: search_products print("\nTest 1: search_products('tent')") result =
plugin.search_products("tent") count = result.count("[P]") if count == 3: print(f" ? Passed
? found 3 tents") else: print(f" ? Failed ? expected 3 tents, got: {repr(result[:100])}")
all_passed = False
# Test 2: get_cart_total print("\nTest 2: get_cart_total()") result =
plugin.get_cart_total() if result == "$339.98": print(f" ? Passed ? cart total = {result}")
else: print(f" ? Failed ? expected $339.98, got: {result}") all_passed = False
# Test 3: calculate_price_with_tax print("\nTest 3: calculate_price_with_tax('P001',
0.08)") result = plugin.calculate_price_with_tax("P001", 0.08) if result == "$269.99":
print(f" ? Passed ? price with tax = {result}") else: print(f" ? Failed ? expected $269.99,
got: {result}") all_passed = False
print() if all_passed: print("? All tests passed! Your plugin is bug-free.") else: print("Some
tests failed. Review the BUG comments and try again.")
return all_passed
???????????????????????????????????????????????????????????????????????????????? SK
integration test (uses GitHub Models ? requires GITHUB_TOKEN)
???????????????????????????????????????????????????????????????????????????????? async def
run_sk_integration_test(): """Run a full SK agent invocation to verify the plugin is
discoverable.""" print("\n--- SK Integration Test (requires GITHUB_TOKEN) ---")
kernel = sk.Kernel() kernel.add_service( OpenAIChatCompletion( ai_model_id="gpt-4o-mini",
api_key=os.environ["GITHUB_TOKEN"], base_url="https://models.inference.ai.azure.com", ) )
plugin = kernel.add_plugin(OutdoorGearPlugin(), plugin_name="OutdoorGear")
# List what SK discovered discovered = [name for name in plugin.functions] print(f"SK
discovered {len(discovered)} functions: {discovered}")
if "search_products" not in discovered: print(" ?? search_products not discovered ? did you
add @kernel_function? (Bug #1)") else: print(" ? search_products is registered in the
kernel")
if __name__ == "__main__": # Unit tests first (no API calls needed) unit_pass =
run_unit_tests()
# SK integration test (tests plugin discovery) if os.environ.get("GITHUB_TOKEN"):
asyncio.run(run_sk_integration_test()) else: print("\n(Skipping SK integration test ? set
GITHUB_TOKEN to enable)")
```

Supporting file: scenario.json

```
{ "lab": "023", "steps": [ "define_tools", "bound_memory", "route_explicitly", "red_team"
], "required_route": "local", "credentials_required": false, "real_path": { "optional":
true, "cost_cap_required": true, "cleanup_required": true } }
```

Supporting file: test_scenario.py

```
import json import unittest from pathlib import Path
class Phase3ScenarioTests(unittest.TestCase): @classmethod def setUpClass(cls): cls.data =
json.loads(Path("scenario.json").read_text(encoding="utf-8"))
def test_identity_and_steps(self): self.assertEqual(self.data["lab"], "023")
self.assertEqual(len(self.data["steps"]), 4)
def test_free_first_contract(self): self.assertEqual(self.data["required_route"], "local")
self.assertFalse(self.data["credentials_required"])
def test_real_path_guardrails(self): real = self.data["real_path"]
self.assertTrue(real["optional"]) self.assertTrue(real["cost_cap_required"])
self.assertTrue(real["cleanup_required"])
if __name__ == "__main__": unittest.main()
```