

Lab 014: Hello Agent avec Microsoft Agent Framework 1.0

Level: L100 Parcours: ai-agents Durée: ~60 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

Créer un agent MAF 1.0 avec un adaptateur local Séparer instructions, modèle, outils et

exécution Migrer un agent Semantic Kernel minimal sans planners Tester le repli déterministe

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: inspect_contract

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: implement_agent

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: add_tool

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: validate_trace

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

```
ai_model_id="gpt-4o-mini" # api_key=os.environ["GITHUB_TOKEN"] #
```

```
base_url="https://models.inference.ai.azure.com" # Return the kernel raise
NotImplementedError("TODO 2-3: implement create_kernel()")
???????????????????????????????????????????????????????????????????????????????????? Exercise 2:
Add a semantic (prompt-based) function
????????????????????????????????????????????????????????????????????????????????????
GEAR_ADVISOR_PROMPT = """ You are an expert outdoor gear advisor for OutdoorGear Inc. Given
a customer's activity and conditions, recommend the right gear.
Activity: {{$activity}} Conditions: {{$conditions}} Budget: {{$budget}}
Provide a specific recommendation with brief reasoning (2-3 sentences). """
async def get_gear_recommendation(kernel, activity: str, conditions: str, budget: str) ->
str: """Use a semantic function to get a gear recommendation.""" # TODO 4: Create a prompt
function from GEAR_ADVISOR_PROMPT using: # kernel.add_function(prompt=...,
plugin_name="GearAdvisor", function_name="recommend") # TODO 5: Invoke the function with
the three parameters # TODO 6: Return the result as a string raise
NotImplementedError("TODO 4-6: implement get_gear_recommendation()")
???????????????????????????????????????????????????????????????????????????????????? Exercise 3:
Add a native (Python) function / plugin
????????????????????????????????????????????????????????????????????????????????????
TODO 7: Import kernel_function decorator from semantic_kernel.functions import
kernel_function
class OutdoorGearPlugin: """A native SK plugin with OutdoorGear business logic."""
# TODO 8: Decorate this method with @kernel_function # Add name="get_gear_category" and
description="Returns gear category for an activity" def get_gear_category(self, activity:
str) -> str: """Return the gear category for a given activity.""" categories = { "hiking":
"Trail & Trekking", "camping": "Camp & Basecamp", "climbing": "Climb & Mountaineer",
"kayaking": "Water Sports", "skiing": "Snow & Ice", "cycling": "Bike & Ride", }
activity_lower = activity.lower() for key, category in categories.items(): if key in
activity_lower: return category return "General Outdoor"
# TODO 9: Decorate this method with @kernel_function # Add name="get_budget_tier" and
description="Returns budget tier label" def get_budget_tier(self, budget_usd: int) -> str:
"""Return budget tier based on dollar amount.""" if budget_usd < 100: return "Budget" elif
budget_usd < 300: return "Mid-range" elif budget_usd < 700: return "Premium" else: return
"Professional"
async def use_native_plugin(kernel): """Add the native plugin to the kernel and invoke its
functions.""" # TODO 10: Add OutdoorGearPlugin() to the kernel: #
kernel.add_plugin(OutdoorGearPlugin(), plugin_name="OutdoorGear") # TODO 11: Invoke
get_gear_category with activity="rock climbing" # TODO 12: Invoke get_budget_tier with
budget_usd=250 # Print both results raise NotImplementedError("TODO 10-12: implement
use_native_plugin()")
???????????????????????????????????????????????????????????????????????????????????? Exercise 4:
Simple chat loop with history ??????????????????????????????????????????????????????????????????
???????????????????? AGENT_SYSTEM = """You are Gear, an AI shopping assistant for OutdoorGear
Inc. Help customers find the perfect gear for their adventures. Keep responses to 2-3
sentences. End each response with one follow-up question."""
async def chat_loop(kernel): """A simple multi-turn chat using ChatHistory.""" # TODO 13:
Create a ChatHistory() instance # TODO 14: Add the AGENT_SYSTEM string as a system message
# TODO 15: Get the chat service from kernel: kernel.get_service() # TODO 16: Loop: read
user input ? add to history ? invoke ? print ? repeat print("TODO: implement chat_loop()")
```

```

???????????????????????????????????????????????????????????????????????????????? Main
???????????????????????????????????????????????????????????????????????????????? async def
main(): print("=== Exercise 1: Create Kernel ===") # kernel = create_kernel() # print("?
Kernel created successfully!")
print("\n=== Exercise 2: Semantic Function ===") # result = await get_gear_recommendation(
# kernel, # activity="winter mountaineering", # conditions="-15°C, wind, possible snow", #
budget="$400" # ) # print(result)
print("\n=== Exercise 3: Native Plugin ===") # await use_native_plugin(kernel)
print("\n=== Exercise 4: Chat Loop ===") # await chat_loop(kernel)
print("\n? Complete the TODOs above and uncomment the exercises!")
if __name__ == "__main__": asyncio.run(main())

```

Supporting file: requirements.txt
semantic-kernel>=1.0.0 openai>=1.0.0

Supporting file: scenario.json

```

{ "lab": "014", "steps": [ "inspect_contract", "implement_agent", "add_tool",
"validate_trace" ], "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"], "014")
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()

```