Semantic Kernel Agent Demos Overview

# Step 1: Basic Chat Completion Agent

This file (`step1\_chat\_completion\_agent\_simple.py`) is the most basic implementation of a Semantic Kernel agent using Azure OpenAI.  
  
Core Features:  
- Utilizes `ChatCompletionAgent` from Semantic Kernel.  
- Connects to AzureChatCompletion service using default environment variables or inline arguments.  
- Sends a list of `USER\_INPUTS` in a loop, invoking the agent's `get\_response` method.  
  
Code Sections:  
1. \*\*Agent Setup\*\*: Initializes a `ChatCompletionAgent` with a name ("Assistant") and basic instructions (e.g., “Answer questions about the world in one sentence.”).  
2. \*\*Response Loop\*\*: Iterates over a list of hardcoded inputs and prints the agent's responses.  
  
Why it’s cool:  
- A perfect starter file for learning how Semantic Kernel integrates with Azure OpenAI.  
- Clear structure: input → response.  
- Easy to modify and extend for basic Q&A or chatbot scenarios.

# Step 4: Agent with Simple Plugin (No Kernel)

This file (`step4\_chat\_completion\_agent\_plugin\_simple.py`) extends Step 1 by demonstrating the use of a Semantic Kernel plugin (function/tool calling), without using the `Kernel` class explicitly.  
  
Core Features:  
- Introduces a `MenuPlugin` with two mock functions: `get\_specials` and `get\_item\_price`.  
- These functions are decorated with `@kernel\_function` and registered inline.  
- The agent can now interpret user questions like "What is the special soup?" and invoke the correct function automatically.  
  
Code Sections:  
1. \*\*Plugin Definition\*\*: `MenuPlugin` defines static outputs for demonstration purposes.  
2. \*\*Agent Setup\*\*: Agent is initialized with the plugin embedded directly, rather than through a Kernel.  
3. \*\*Auto Tool Invocation\*\*: The agent detects when a question maps to a registered function and executes it.  
  
Why it’s cool:  
- Demonstrates how Semantic Kernel tools/plugins work.  
- Allows “agentic” behavior with function calling without needing a full Kernel setup.  
- Great for prototyping product catalogs, FAQs, or restaurant menus.

# Step 5: Full Kernel Integration with Real-Time Weather Plugin

This file (`step5\_chat\_completion\_agent\_plugin\_with\_kernel.py`) is a full end-to-end example of a functional AI agent using the complete Semantic Kernel architecture, including:  
  
- `Kernel()` object to register plugins and AI services.  
- Function-based plugin using live API calls to WeatherAPI.com.  
- Auto tool invocation enabled with `FunctionChoiceBehavior.Auto()`.  
- Threaded conversation management using `ChatHistoryAgentThread`.  
  
Code Sections:  
1. \*\*Weather Plugin\*\*: The `WeatherPlugin` defines `get\_weather(city)` which sends a real HTTP request to the WeatherAPI.com `current.json` endpoint and parses the response.  
2. \*\*Kernel Setup\*\*: Adds the plugin and connects to the Azure ChatCompletion service using your provided deployment, endpoint, and API key.  
3. \*\*Prompt Execution Settings\*\*: Fetches the execution settings for the chat model and configures tool-calling behavior.  
4. \*\*Agent Configuration\*\*: Creates a `ChatCompletionAgent` and passes the kernel and settings via `KernelArguments`.  
5. \*\*Conversation Threading\*\*: Stores conversation context across multiple user inputs using `ChatHistoryAgentThread`, which mimics real conversational memory.  
  
Why it’s cool:  
- Demonstrates real-time, dynamic agent behavior using live data.  
- Uses Semantic Kernel’s full capabilities: service registration, plugin injection, function invocation, and memory threading.  
- A great starting point for building retail, travel, or assistant agents that respond based on external APIs.  
- Fully modular: you can add more plugins (like trip recommendations, packing lists, etc.) without touching core logic.