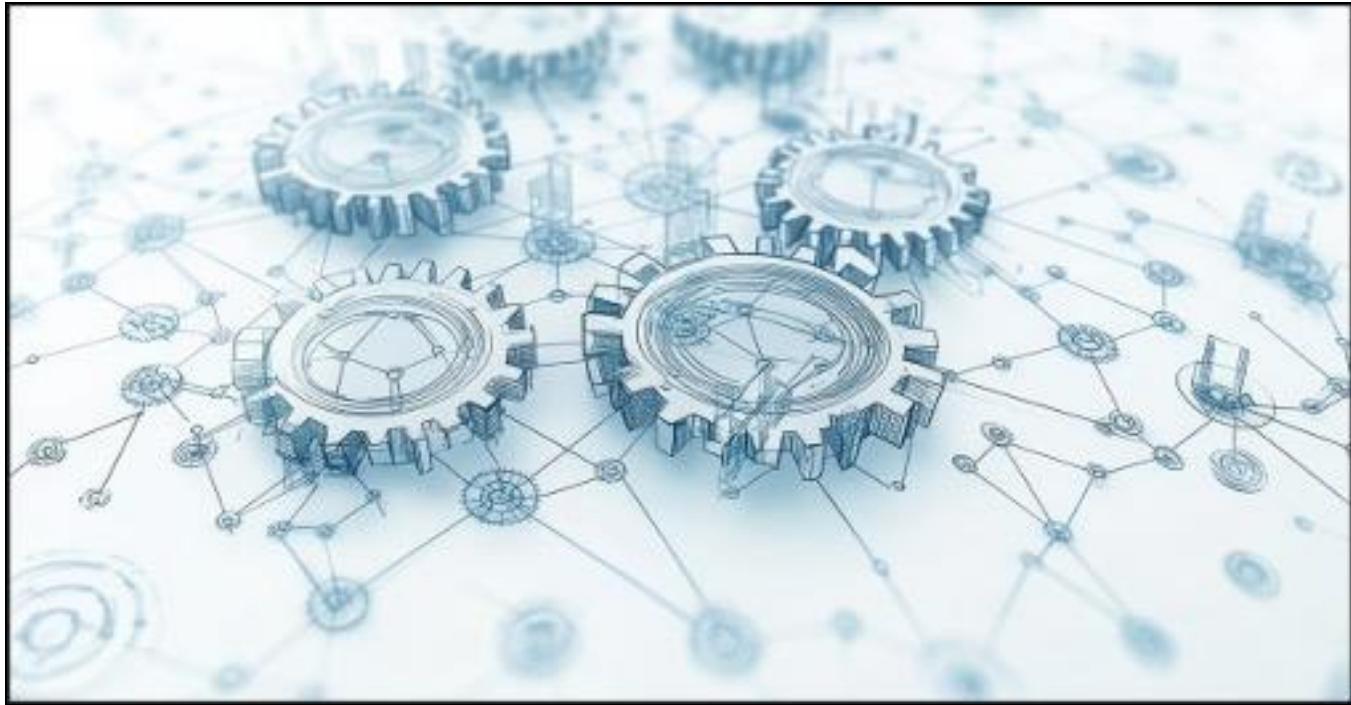


Level 4

Watsonx Orchestrate

Hands-on lab guide – Create and improve agents with copilot



Laurent Tarin
Laurent.tarin@fr.ibm.com
Learning Content Development, Data & AI

Version 1.0

Sept 2025

LEGAL NOTICES

This information was developed for products and services offered in the USA. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119 Armonk, NY 10504-1785
United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice. Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All names and references for organizations and other business institutions used in this deliverable's scenarios are fictional. Any match with real organizations or institutions is coincidental. All names and associated information for people in this deliverable's scenarios are fictional. Any match with a real person is coincidental.

TRADEMARKS

IBM, the IBM logo, and [ibm.com](#) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at <https://www.ibm.com/legal/copyright-trademark>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates. INFOBLOX is a trademark of Infoblox Inc. or its affiliated companies, registered in the United States and other countries.

Microsoft®, Azure, Windows, Windows NT, Windows Server, Microsoft® Excel®, Microsoft® Word®, Microsoft® Outlook®, Microsoft® Excel® logo, Microsoft® Word® logo, Microsoft® Outlook® logo, Microsoft® Windows App, Microsoft® Windows App logo and the Windows logo are trademarks of Microsoft® Corporation in the United States, other countries, or both.

Salesforce and Salesforce logo are registered trademarks of SALESFORCE, INC. (CORPORATION; CALIFORNIA, USA).

Slack is a registered trademark of Salesforce, Inc (CORPORATION; DELAWARE, USA).

Red Hat®, JBoss®, OpenShift®, Fedora®, Hibernate®, Ansible®, CloudForms®, RHCA®, RHCE®, RHCSA®, Ceph®, and Gluster® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Google, Google logo, Google Tasks, Google Cloud Platform, GMAIL, GMAIL logo are registered trademarks of GOOGLE LLC and this material is not endorsed by or affiliated with Google in any way.

Elasticsearch, Elastic search logo, Elastic, Elastic logo are registered trademarks of Elasticsearch BV (LIMITED LIABILITY COMPANY; NETHERLANDS) in United States, other countries or both.

Milvus, Milvus logo are registered trademarks of LF Projects, LLC (LIMITED LIABILITY COMPANY; DELAWARE, USA) in United States, other countries or both.

Meta, Meta logos, Llama, Llama logos are registered trademarks of META PLATFORMS, INC. (CORPORATION; California, USA) in United States, other countries or both.

Mistral, Mistral logo, Mistral AI, Mistral AI logo are registered trademarks of Mistral AI (SOCIÉTÉ PAR ACTIONS SIMPLIFIÉE (SAS); FRANCE) in United States, other countries or both.

PyCharm, PyCharm logo are registered trademarks of JetBrains s.r.o. (CORPORATION; CZECH REPUBLIC) in United States, other countries or both.

Bruno is a trademark held by Anoop M D. Bruno logo is sourced from OpenMoji. License: CC BY-SA 4.0

Apple, Apple logo, Apple store, App store, App store logo are registered trademarks of Apple Inc. (CORPORATION; CALIFORNIA, USA) in United States, other countries or both.

VMWARE, VMWARE logos are registered trademarks of VMware LLC (LIMITED LIABILITY COMPANY; Delaware, USA) in United States, other countries or both.

RANCHER, RANCHER logo, Rancher Desktop, Rancher Desktop logo are registered trademarks of SUSE LLC (LIMITED LIABILITY COMPANY; MASSACHUSETTS, USA) in United States, other countries or both.

Langfuse, Langfuse logo are registered trademarks of Langfuse GmbH (GESELLSCHAFT MIT BESCHRÄNKTER HAFTUNG (GMBH); Germany, Germany) in United States, other countries or both.

OpenMeteo is a registered trademark owned by Patrick Zippenfenig (Individual, Germany) in United States, other countries or both.

© Copyright International Business Machines Corporation 2025. This document may not be reproduced in whole or in part without the prior written permission of IBM.

Table of Contents

Introduction	6
About this lab	6
Environments	6
How to get support	7
Prerequisites	7
Warning	7
Prepare your environment	8
Reset the server.....	10
Import the required tools.....	12
Create a copilot working folder.....	14
Start Copilot server.....	16
Section 1: Create a simple agent with copilot	18
Create the agent from scratch	18
Refine the agent.....	22
Review the generated agent	24
Preview your agent	26
Section 2: Refine an existing agent with copilot	30
Initiate the refining process	30
Review the improved agent.....	33
Preview the agent	34
Section 3: Create a complex Agentic AI agent with copilot.....	37
Part 1: Create a Life insurance agent.....	37
Start copilot to create the agent.....	37
Refine the prompt	40
Review the agent	41
Part 2: Create a Monthly payment agent.....	43
Start copilot to create the agent.....	43
Refine the prompt	46
Review the agent	47
Part 3: Create a managing agent with collaborators	48
Import the new agents	49
Start copilot to create the agent.....	51
Refine the prompt	54
Review the agent	55

Preview the agent.....	59
Summary.....	62

Introduction

About this lab

This lab will teach you how to use the watsonx Orchestrate copilot (Copilot from IBM. Nothing to relate to other vendors) feature to create your own agents from scratch, improve existing ones or progressively build complex agentic AI solutions involving distinct tools and agents.

This lab is composed of 3 independent sections that we recommend performing in the following sequence:

- Section 1: Create a simple agent from scratch
- Section 2: Refine an existing agent
- Section 3: Create an advanced AI agents

At the end of this lab, you will be able to:

- Create an agent from scratch using a chat-based interface.
- Improve existing agents prompts and instructions using copilot.
- Leverage existing tools and agents and ask copilot to create even more capable agents.

Environments

This lab has been created and tested using ADK VMware image detailed in the preparation guide.

How to get support

- IBMers can use the **#ba-techlcd-support** Slack channel.
- If you are an IBM Business Partner and require assistance, please open a support case at IBM Technology Zone Help.

Prerequisites

- You have completed the following labs:
 - Completed the wxO L4 Preparation guide

You have completed the ADK – Creating tools and agents

You have completed the ADK - Collaborating agents with Python tools and Langfuse

Warning

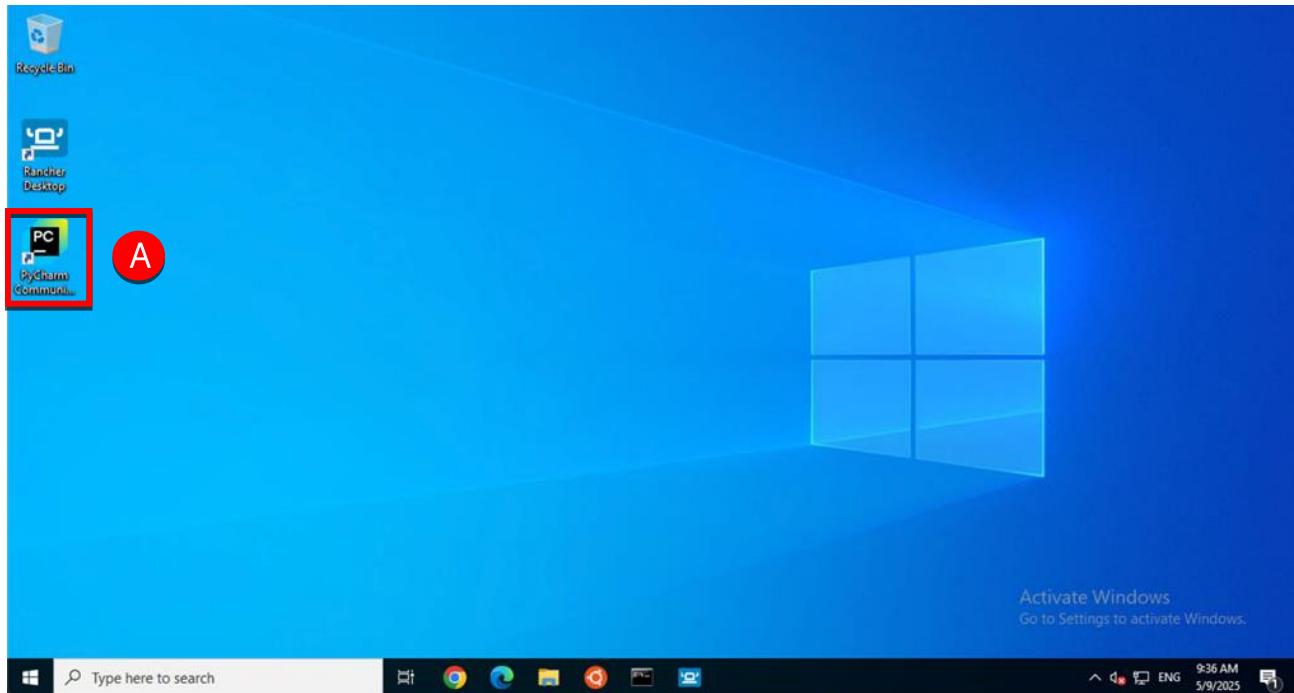
LLMs that have been used to create the lab may evolve and change rapidly. It is very likely that you will use a different LLM version that may produce slightly different results compared to what is shown in the screen captures. Just move on with the lab and try to adapt the prompts/instructions accordingly based on the responses from copilot.

Due to the non-deterministic nature of LLMs, you will have to adapt your answers based on the responses from copilot. Use the examples provided in this guide but be prepared to adapt your input and choices based on the responses from copilot.

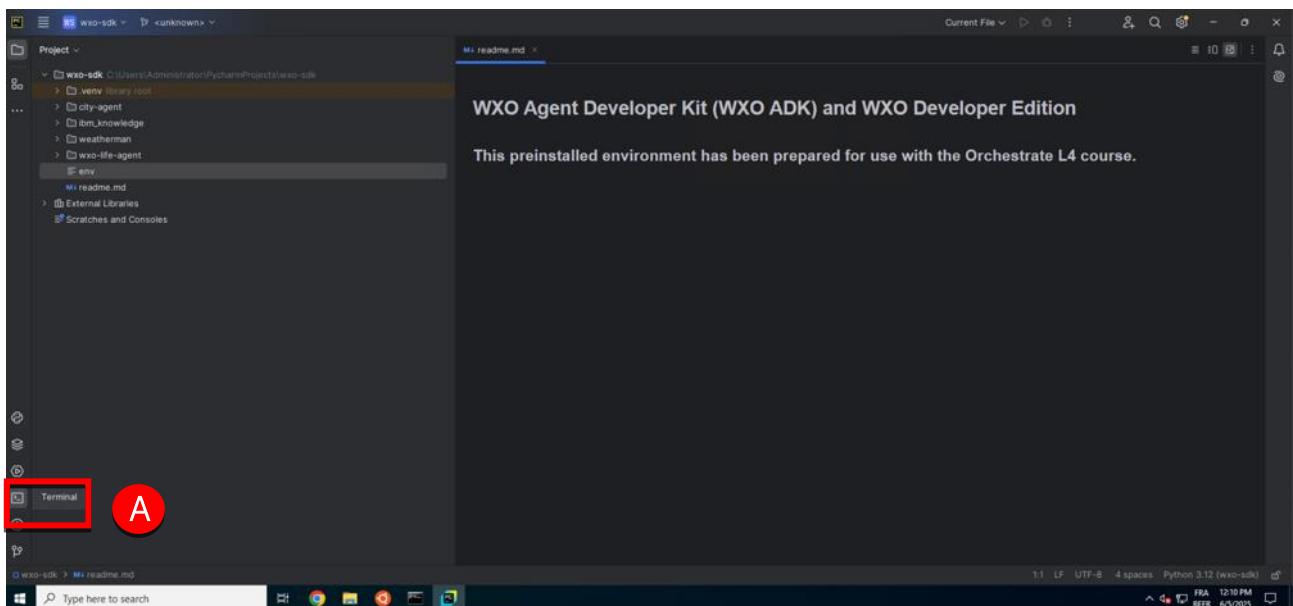
Prepare your environment

In this section, you will prepare the ADK environment to work with copilot. In order not to confuse copilot with the other agents you may have already created during this learning plan, you will reset the server to clean it up and just re-import the required tools to start from a clean base.

1. Double-click the **PyCharm** icon (A), to open the editor. (PyCharm may be open already if you have used this environment to complete previous labs.)

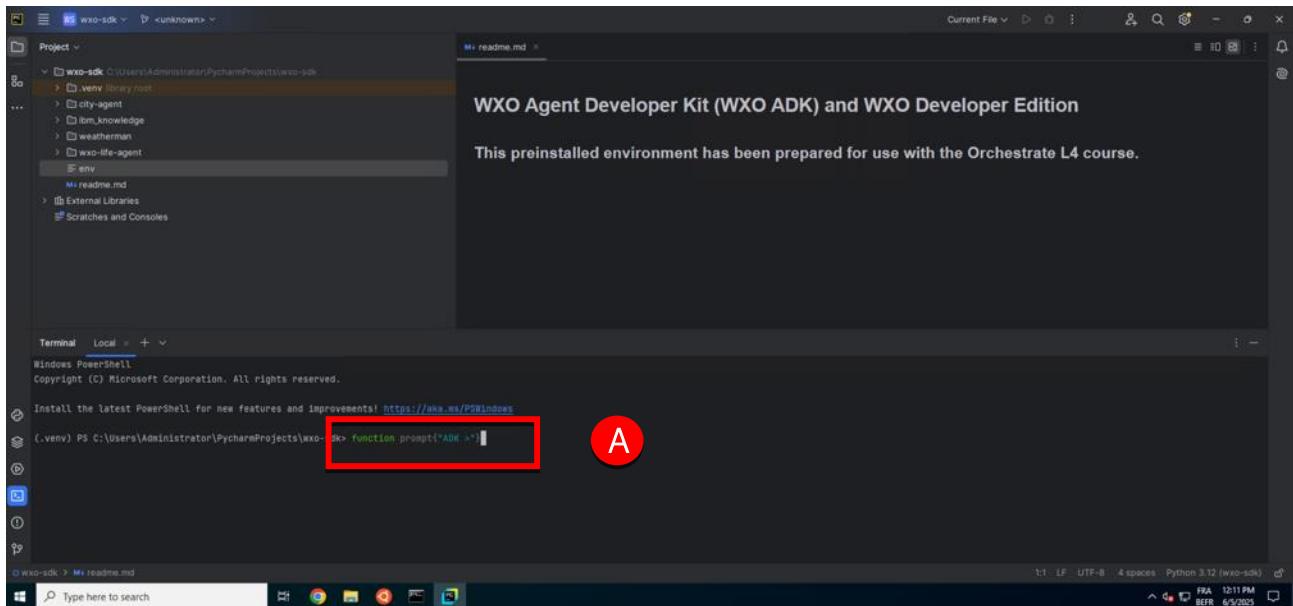


2. Click the terminal icon (A), to open the terminal.



3. Enter the following command into the terminal (A), then press Enter:

```
function prompt{"ADK >"}
```



This changes the prompt and will improve the readability of commands used in this lab.

Reset the server

1. Enter the following command into the terminal to stop your chat service, then press Enter.

```
orchestrate chat stop
```

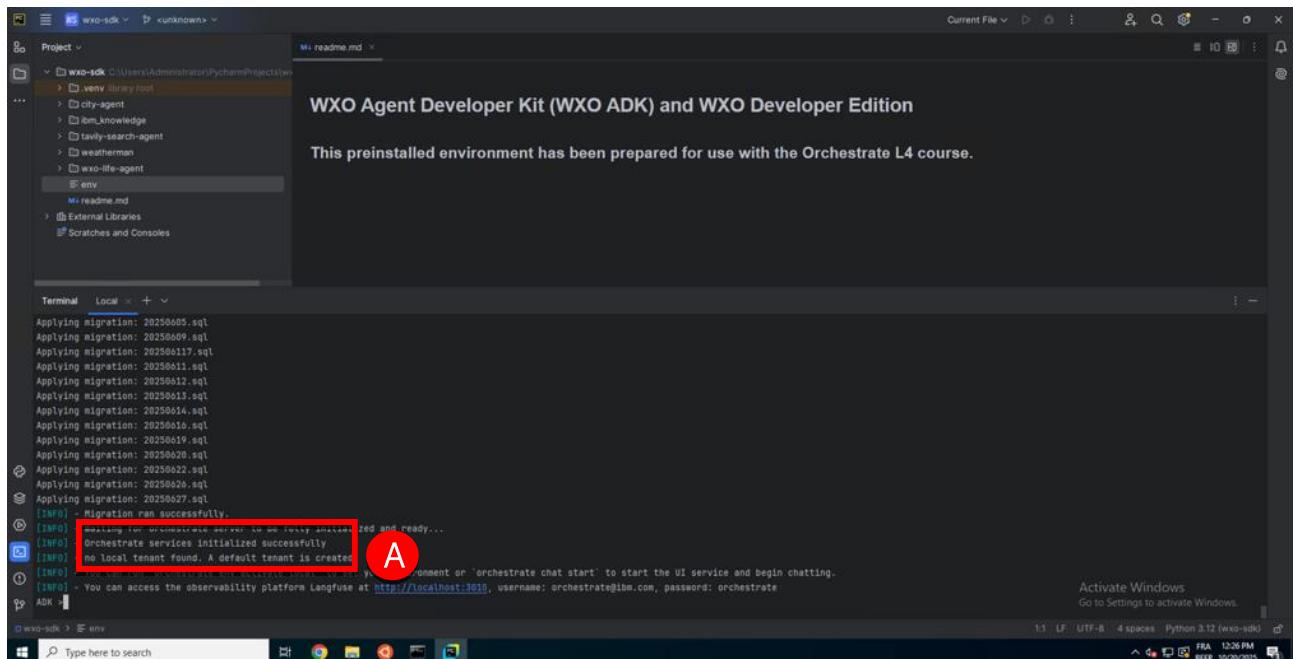
2. Enter the following command into the terminal to reset your server, then press Enter.

```
orchestrate server reset
```

3. Enter the following command into the terminal to restart your server, then press Enter.

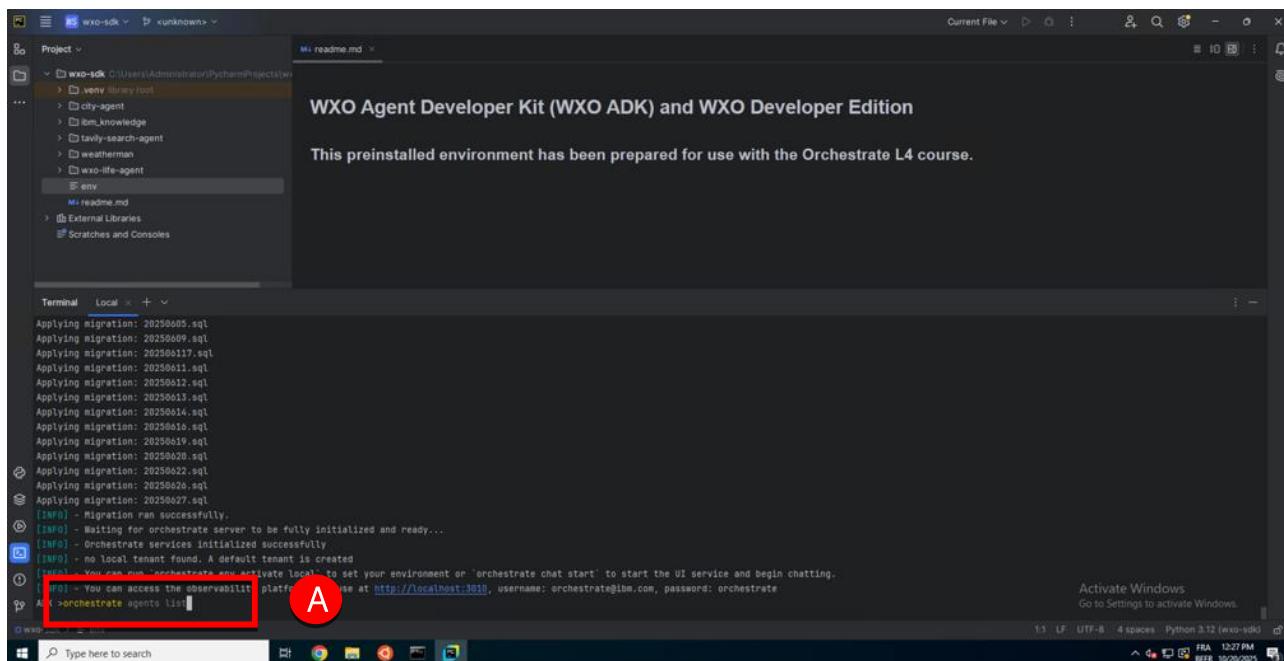
```
orchestrate server start -l -e env
```

4. Wait for your server to be started (A).



5. Type the following command to check that all the agents have been removed.

```
orchestrate agents list
```



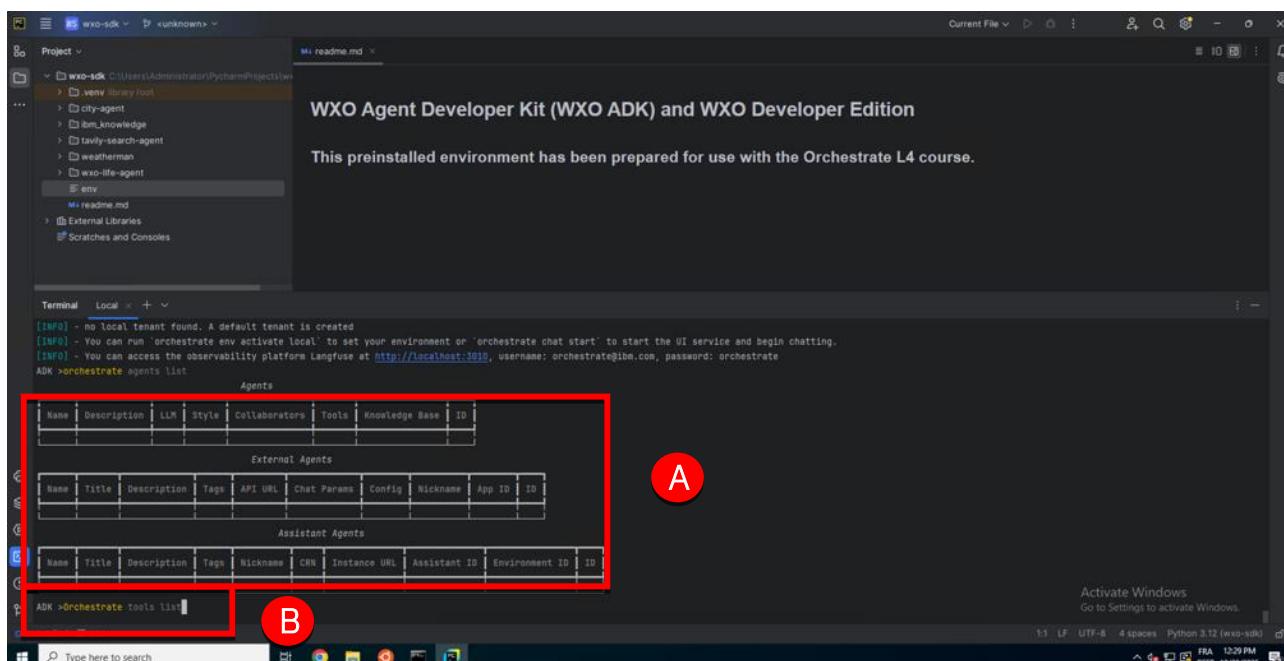
The screenshot shows a terminal window within a development environment. The terminal output displays the application of several SQL migrations from files like 20250605.sql to 20250627.sql. After the migrations, it shows the creation of a default tenant and the initialization of orchestrate services. A command is then run:

```
ADK >orchestrator agents list
```

A red circle labeled 'A' highlights the command in the terminal. The terminal window has a dark theme with light-colored text.

- Check that no agents are listed (A). Type the following command (B) to check that all the tools have been removed.

orchestrator tools list

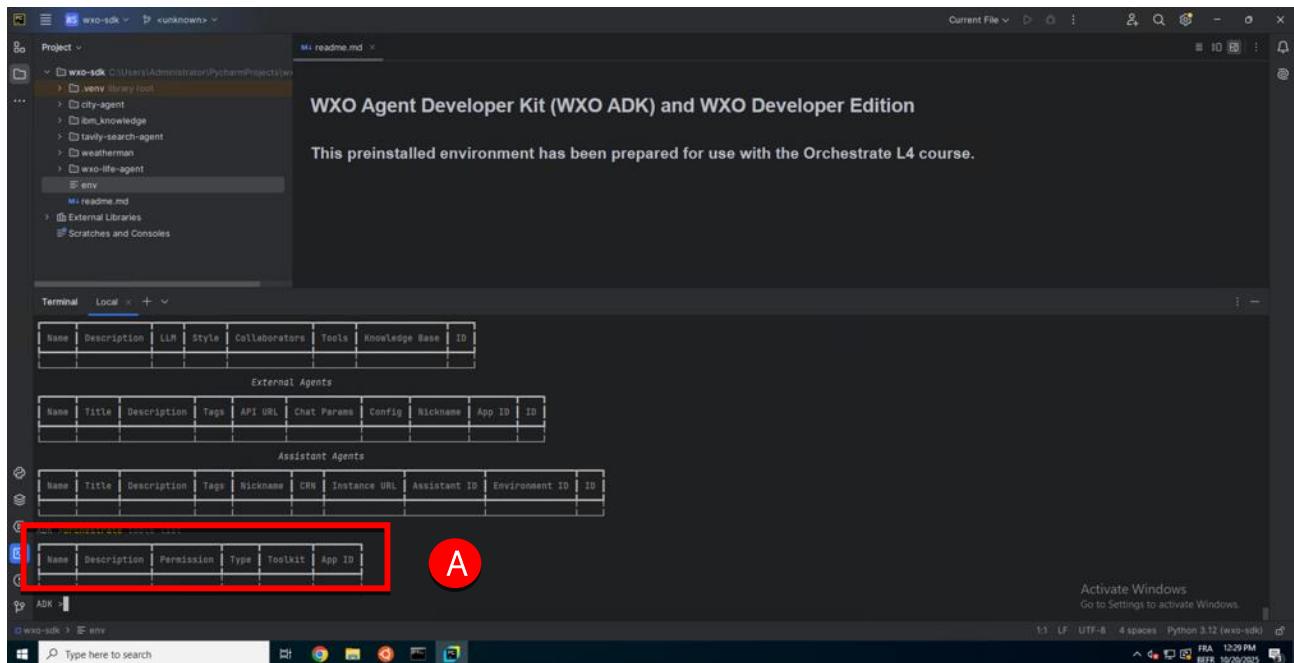


The screenshot shows a terminal window within a development environment. The terminal output shows the command:

```
ADK >Orchestrator tools list
```

A red circle labeled 'A' highlights the command in the terminal. Below the terminal, there is a table titled "Agents" with three sections: "External Agents", "Assistant Agents", and "Assistant Agents". Each section has a header row with columns like Name, Description, Title, Tags, API URL, Chat Params, Config, Nickname, App ID, and ID. A red box labeled 'B' encloses the entire table area.

7. Check that no tools are listed anymore (A).



Your server is now reset. All the tools, agents, environments etc... have been removed. Next you will re-import the required tools to perform this lab.

Import the required tools

1. Run the following commands (A) to import the four required tools:

```
orchestrate tools import -k openapi -f ./weatherman/tools/weather-api.yaml
orchestrate tools import -k python -f wxo-life-agent/tools/life_quote.py
orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
orchestrate tools import -k python -f wxo-life-
agent/tools/monthly_payment_calc.py
```

WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

Terminal Local +

Assistant Agents

Name	Title	Description	Tags	Nickname	CRN	Instance URL	Assistant ID	Environment ID	ID
------	-------	-------------	------	----------	-----	--------------	--------------	----------------	----

```
AOK >Orchestrate.tools list
[Name] [Description] [Permission] [Type] [Toolkit] [App ID]

ADMK >orchestrate tools import -k openapi -f ./weathermap/tools/weather-api.yaml
[INFO] - Tool 'forecast' imported successfully
ADMK >orchestrate tools import -k python -f wko-life-agent/tools/life_quote.py
[INFO] - Tool 'life_quote' imported successfully
ADMK >orchestrate tools import -k python -f wko-life-agent/tools/interest_rates.py
[INFO] - Tool 'interest_rate_lookup' imported successfully
ADMK >orchestrate tools import -k python -f wko-life-agent/tools/monthly_payment_calc.py
[INFO] - Tool 'monthly_payment_calc' imported successfully
ADMK :
```

Activate Windows
Go to Settings to activate Windows.

1:1 LF UTF-8 4 spaces Python 3.12 (wko-sdk) FRA 12:32 PM BEFR 10/20/2025

- Type the following command (B) to check that all the tools have been imported.

orchestrate tools list

Terminal Local +

Assistant Agents

Name	Title	Description	Tags	Nickname	CRN	Instance URL	Assistant ID	Environment ID	ID
------	-------	-------------	------	----------	-----	--------------	--------------	----------------	----

```
AOK >Orchestrate.tools list
[Name] [Description] [Permission] [Type] [Toolkit] [App ID]

ADMK >orchestrate tools import -k openapi -f ./weathermap/tools/weather-api.yaml
[INFO] - Tool 'forecast' imported successfully
ADMK >orchestrate tools import -k python -f wko-life-agent/tools/life_quote.py
[INFO] - Tool 'life_quote' imported successfully
ADMK >orchestrate tools import -k python -f wko-life-agent/tools/interest_rates.py
[INFO] - Tool 'interest_rate_lookup' imported successfully
ADMK >orchestrate tools import -k python -f wko-life-agent/tools/monthly_payment_calc.py
[INFO] - Tool 'monthly_payment_calc' imported successfully
ADMK >orchestrate tools list
ADMK :
```

Activate Windows
Go to Settings to activate Windows.

1:1 LF UTF-8 4 spaces Python 3.12 (wko-sdk) FRA 12:33 PM BEFR 10/20/2025

3. Check that the four required tools are listed (A).

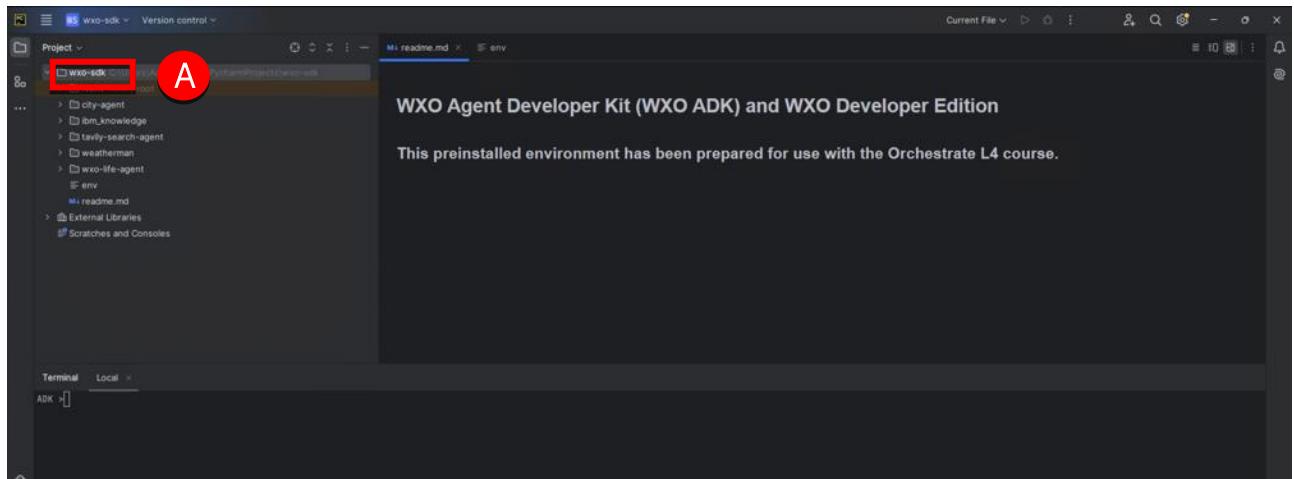
WXO Agent Developer Kit (WXO ADK) and WXO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

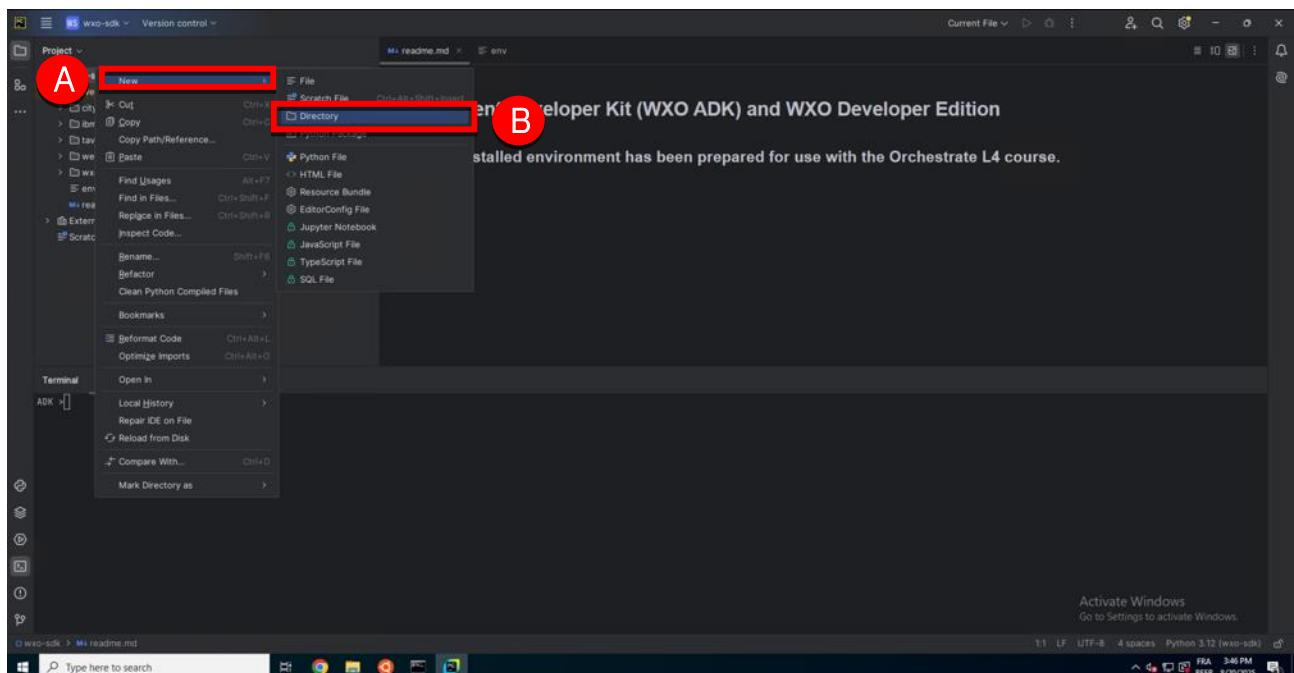
Name	Description	Permission	Type	Toolkit	App ID
forecast	Weather information for given WGS84 latitude and longitude coordinates.	read_only	openapi		
interest_rate_lookup	This tool will determine the interest rate to be used when calculating monthly payments for policies. The rate will be based on the annual premium of the policy.	read_only	python		
life_quote	Calculates a coverage premium for life insurance when provided with amount of cover required and the age of the customer.	read_only	python		
monthly_payment_calc	Calculates the monthly repayment for a policy using the annual policy premium and interest rate.	read_only	python		

Create a copilot working folder

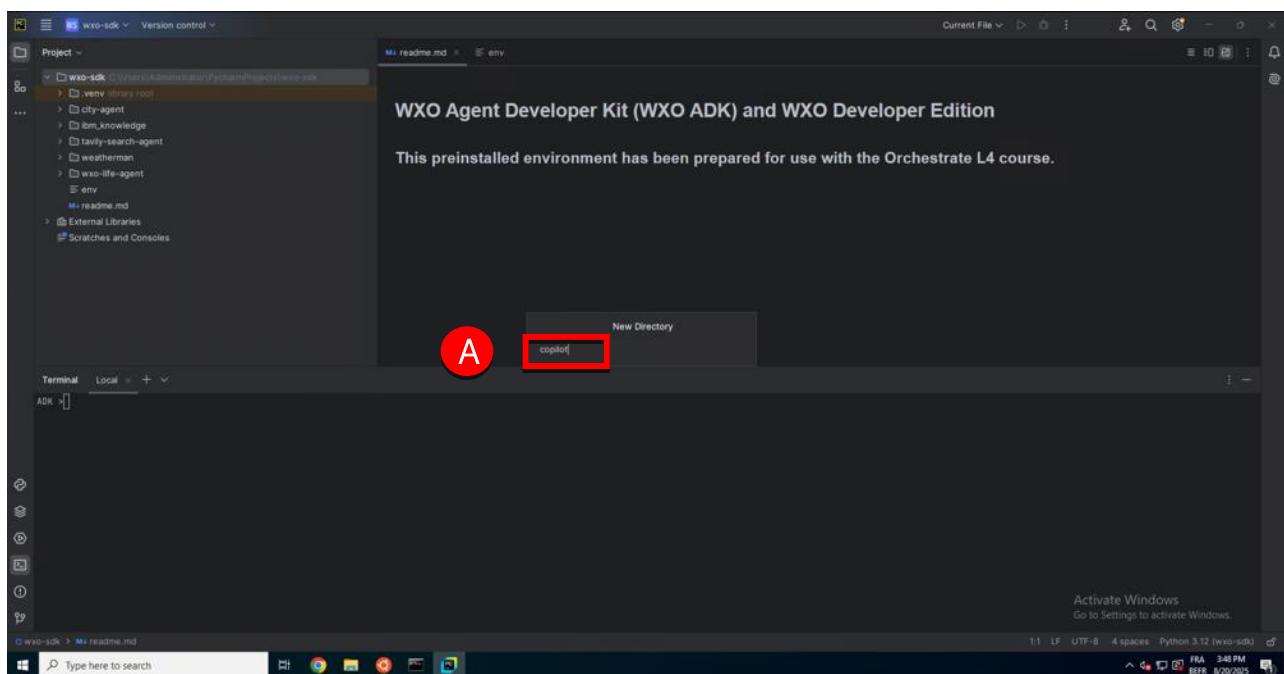
1. Open your ADK VM environment as explained in the Prep Guide.
2. Right-click on the **wxo-sdk** folder (A).



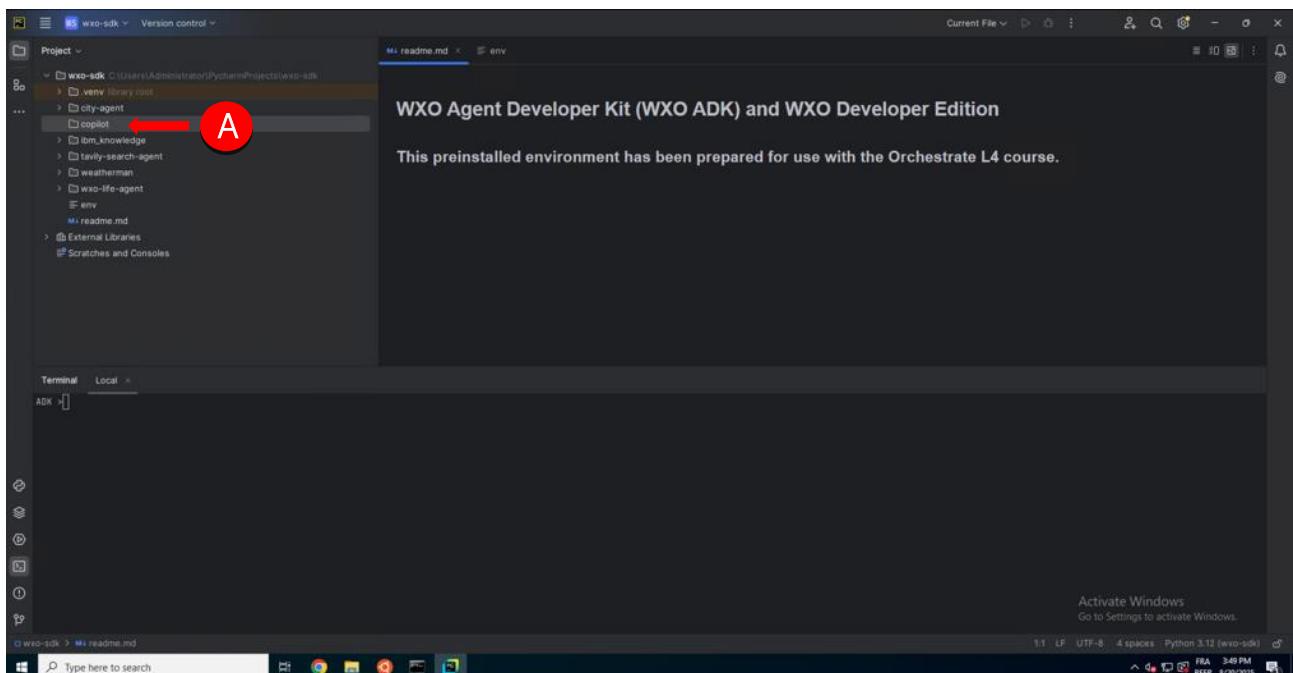
3. Click **New** (A) and then **Directory** (B).



4. Type 'copilot' in the folder New Directory field (A) then press enter.



- Check that the folder is correctly created and at the right place (A).

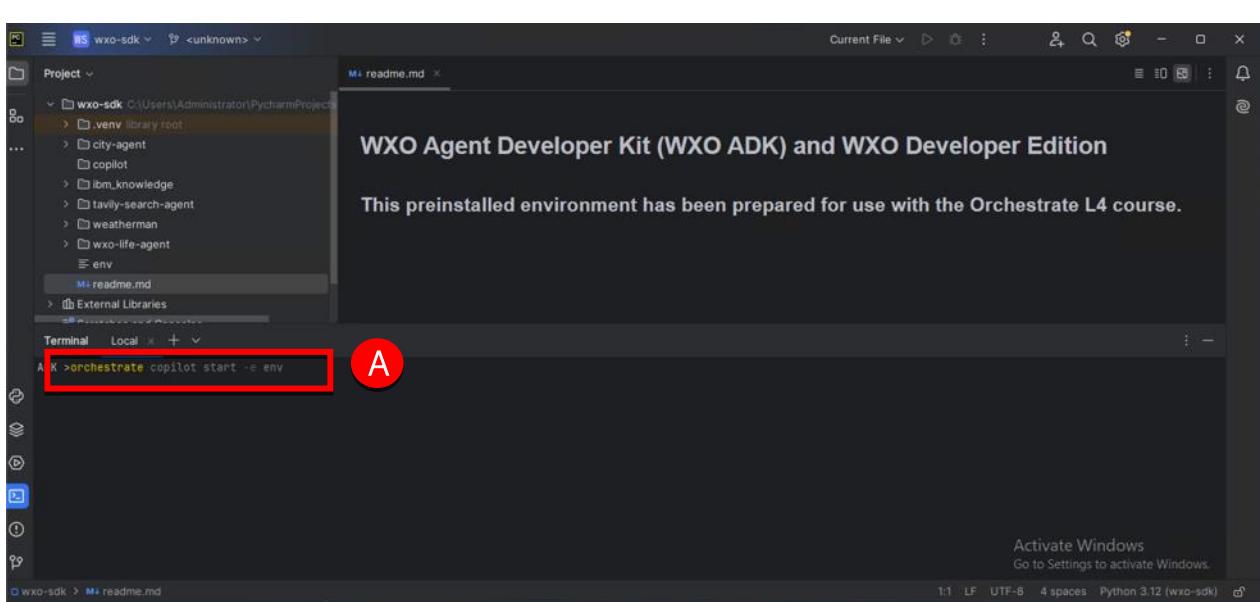


Start Copilot server

Before copilot can be used, the copilot server must be started.

- To start your copilot server, enter the following command into the terminal, then press Enter.

```
orchestrator copilot start -e env
```



Once the copilot server starts you can proceed with this lab.

Note: This guide will provide you some conversation examples with copilot. As copilot leverages LLM reasoning, your own conversation may differ from the ones described in this guide. Just read the copilot questions and provide reasonable responses so you understand the process of using this utility.

This lab focuses on the process for creating agents with Copilot. As Copilot is evolving, the individual steps taken are likely to be different from the examples in this guide.

Section 1: Create a simple agent with copilot

In this first section, you will learn the basics of building an AI Agent prompt from scratch using copilot. Copilot is a CLI utility that generates agent definitions based on your requirements and the tools agents deployed in your environment. The first agent you will create will be simple enough to require no tool, no knowledge, and no collaborators. If you are already familiar with the basics of copilot, you can directly jump to Section 2 where you will use copilot to improve an existing agent.

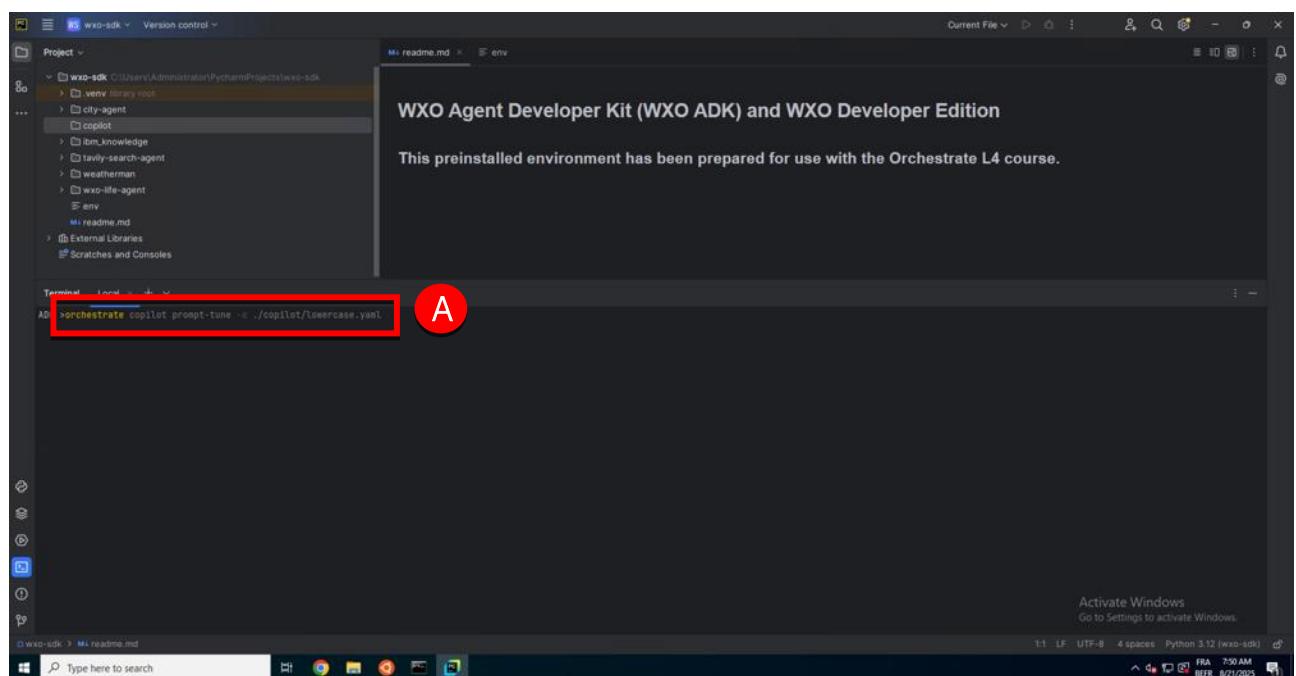
Create the agent from scratch

You are now going to use copilot to create a very simple agent able to lowercase a word or a sentence. You will start from nothing. You will learn the process copilot uses to create a prompt and refine it depending on your needs and on the tools/agents already available in the environment.

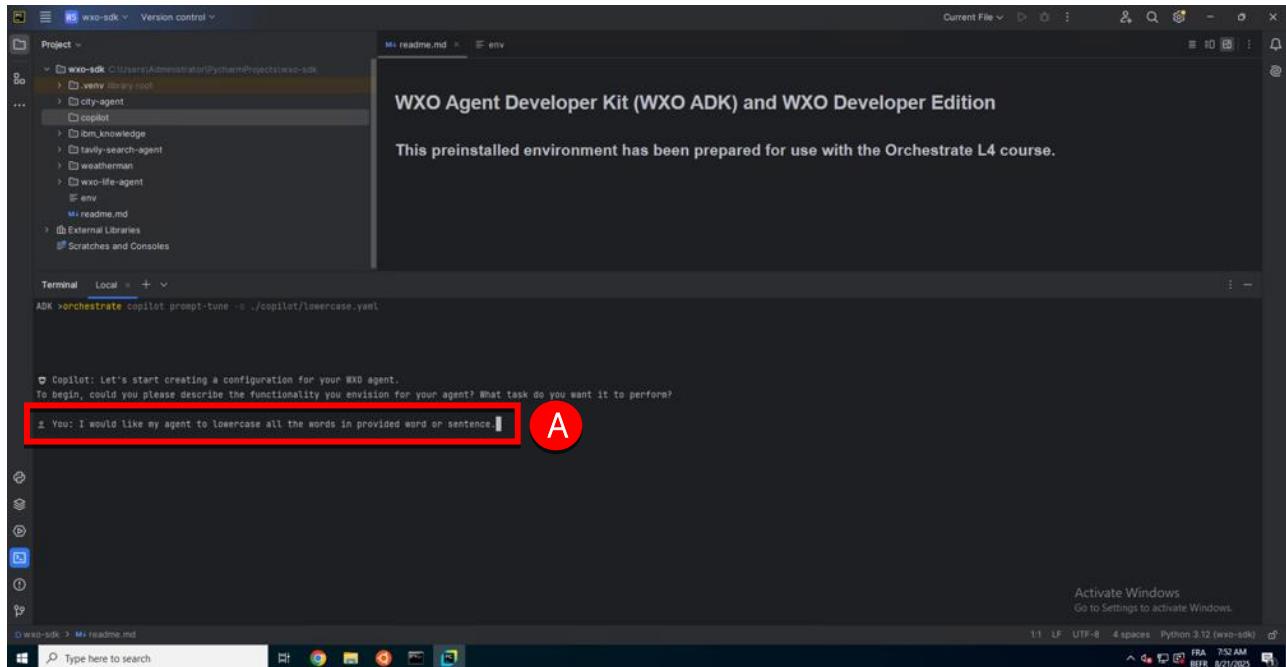
Your lowercase.yaml agent description file will be stored in the copilot folder you have previously created.

1. Start the copilot server typing the following command (A):

```
orchestrate copilot prompt-tune -o ./copilot/lowercase.yaml
```



2. Type 'I would like my agent to lowercase all the words in provided word or sentence.' When prompted by copilot (A).

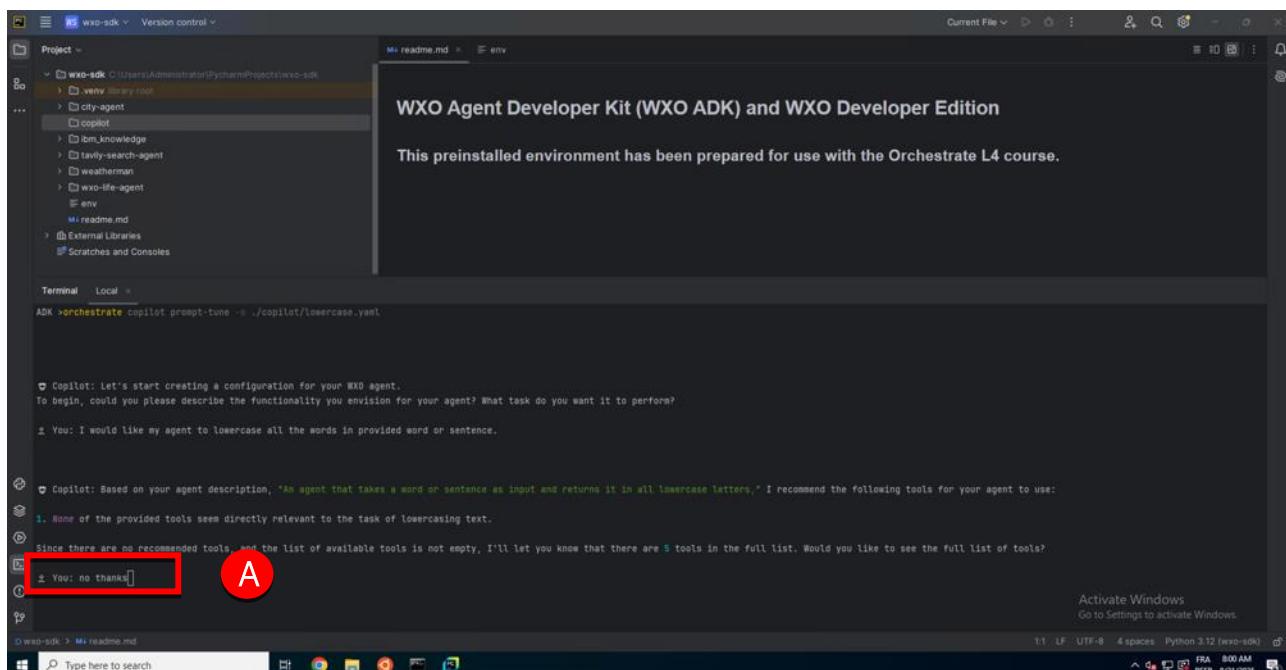


```
ABK >orchestrator copilot prompt-tune -o ./copilot/lowercase.yaml

Copilot: Let's start creating a configuration for your WKO agent.
To begin, could you please describe the functionality you envision for your agent? What task do you want it to perform?
# You: I would like my agent to lowercase all the words in provided word or sentence.
```

After providing you with a first agent description copilot will try to see if tools can help your agent to achieve the task.

3. Let copilot know that you don't want to use the existing tools from the list. In this example we have answered 'No thanks' (A).



```
ABK >orchestrator copilot prompt-tune -o ./copilot/lowercase.yaml

Copilot: Let's start creating a configuration for your WKO agent.
To begin, could you please describe the functionality you envision for your agent? What task do you want it to perform?
# You: I would like my agent to lowercase all the words in provided word or sentence.

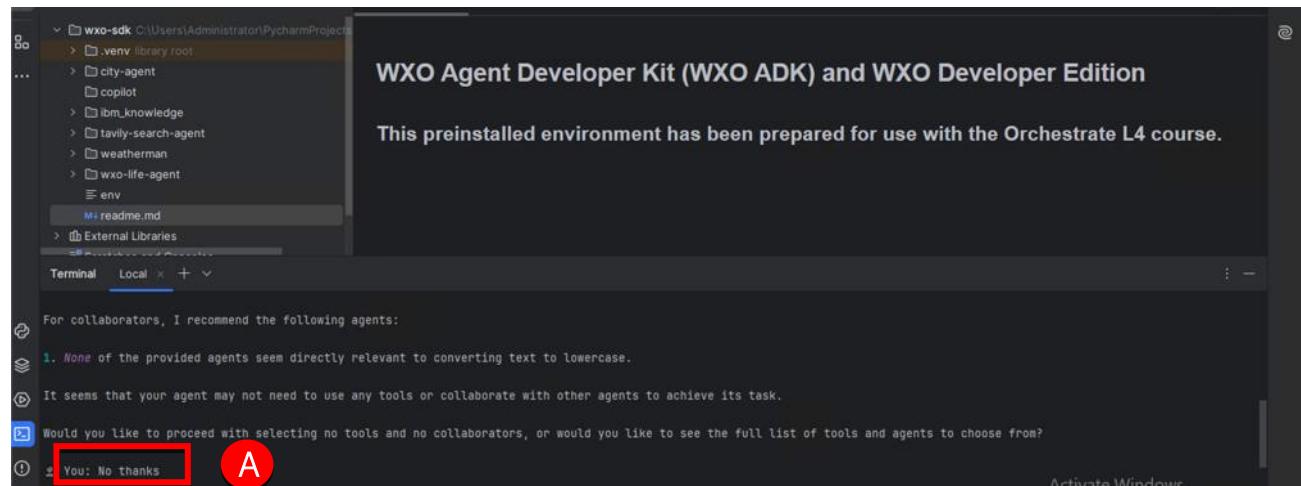
Copilot: Based on your agent description, "An agent that takes a word or sentence as input and returns it in all lowercase letters," I recommend the following tools for your agent to use:
1. None of the provided tools seem directly relevant to the task of lowercasing text.

Since there are no recommended tools, and the list of available tools is not empty, I'll let you know that there are 5 tools in the full list. Would you like to see the full list of tools?
# You: no thanks
```

Next copilot will look for agents to be used as collaborators to help your agent perform the task. Here again, no collaborators are needed.

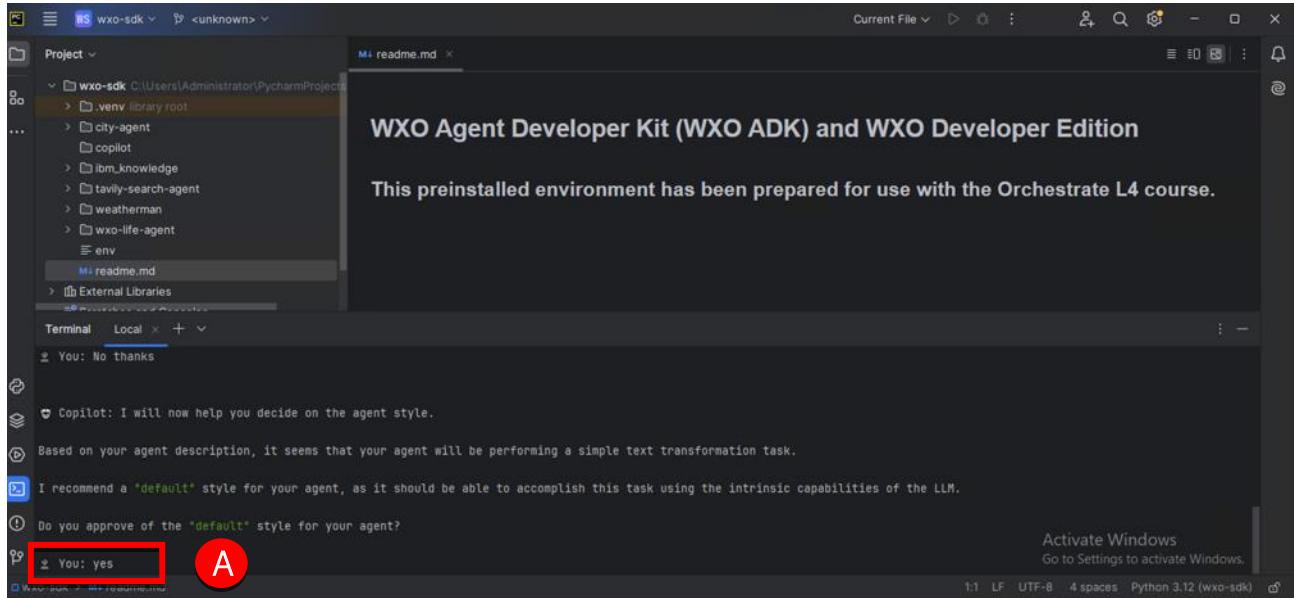
Note: Please remember that this flow is provided as an example. We cannot assume what copilot will ask you as the following step. Just follow your copilot. This lab is about teaching you how to use copilot, not to build a specific agent.

- Let copilot know that you don't want to use collaborators. In this example, we have answered 'No thanks' (A).



Next copilot will help you decide the type of agent to create. It can be a Default or a ReAct (Reason-Act) type of agent if several tools/collaborators are involved. In our simple example, the Default is recommended. Copilot may ask you to decide or approve its choice.

- Let copilot know that a default agent type is good. In our example we have answered yes (A) to approve copilot choice.



Next copilot will ask you about how this agent will be invoked by the users. This will help to improve the prompt. Copilot will prompt you for 3 invocation sentences.

- Type the 3 following sentences (A) one by one when prompted:

```
Can you lowercase the word Laurent
Please remove uppercase letters from Hello World
Lowercase the following ID LT-123455
```

The screenshot shows a PyCharm interface with a terminal window open. The terminal displays the following text:

```
but it might be redundant since your agent is intended to perform the same task.  
Would you like to select any collaborators from the list of available agents?  
± You: no thanks  
  
Copilot: Now that we have discussed the tools and collaborators for your agent, let's move on to deciding on the agent style.  
Based on your agent description, "An agent that takes a word or sentence as input and returns it in all lowercase letters," I suggest that a "default" style would be suitable for your agent. This is because the task of  
lowercasing text is a simple, straightforward operation that doesn't require complex reasoning or multi-step actions.  
Do you approve of the "default" style for your agent?  
± You: yes  
  
Can you lowercase the word Laurent  
Please remove uppercase letters from Hello World  
Lowercase the following ID LT-123455
```

A red box highlights the last three lines of the terminal output, and a red circle labeled 'A' is positioned to the right of the terminal window.

Refine the agent

Copilot will ask further questions to refine the prompt, such as how to format the agent output as shown in this example. Let's start the refining process.

1. Let copilot know that you are willing to refine the prompt. In our example we have answered, 'just provide the lowercased sentence with no additional comment' (A).

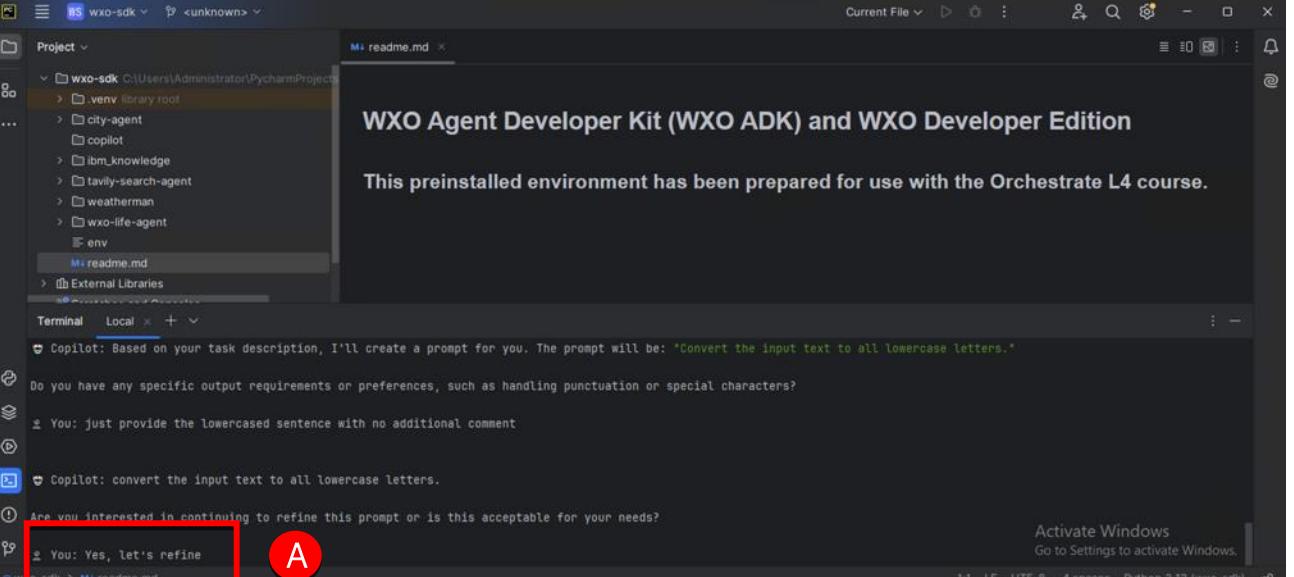
The screenshot shows a PyCharm interface with a terminal window open. The terminal displays the following text:

```
> Lowercase the following ID LT-123455  
[INFO] - You provided the following samples:  
1. Can you lowercase the word Laurent  
2. Please remove uppercase letters from Hello World  
3. Lowercase the following ID LT-123455  
  
Copilot: Based on your task description, I'll create a prompt for you. The prompt will be: "Convert the input text to all lowercase letters."  
Do you have any specific output requirements or preferences, such as handling punctuation or special characters?  
You: just provide the lowercased sentence with no additional comment
```

A red box highlights the last line of the terminal output, and a red circle labeled 'A' is positioned to the right of the terminal window.

Copilot will continue to ask for further refinements.

2. Let Copilot know that you are willing to refine the prompt. In our example we have answered ‘Yes, let’s refine’ (A).



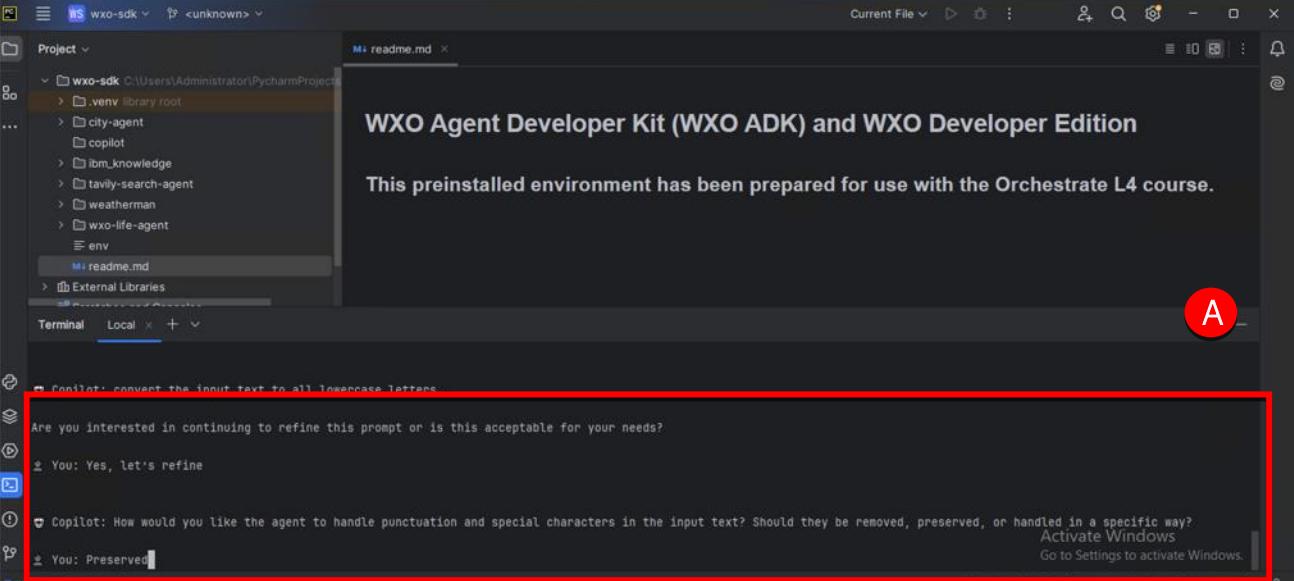
The screenshot shows a PyCharm interface with a terminal window open. The terminal shows a conversation with Copilot:

```

Copilot: Based on your task description, I'll create a prompt for you. The prompt will be: "Convert the input text to all lowercase letters."
Do you have any specific output requirements or preferences, such as handling punctuation or special characters?
You: just provide the lowercased sentence with no additional comment
Copilot: convert the input text to all lowercase letters.
Are you interested in continuing to refine this prompt or is this acceptable for your needs?
You: Yes, let's refine
  
```

A red box highlights the user's response "You: Yes, let's refine". A red circle with the letter "A" is positioned to the right of the terminal window.

3. Copilot will continue to suggest improvements or ask you to continue or stop the refinement process. Here is an example of refinement conversation we have had with copilot (A). Follow your own conversation and let copilot know when you are satisfied (B).



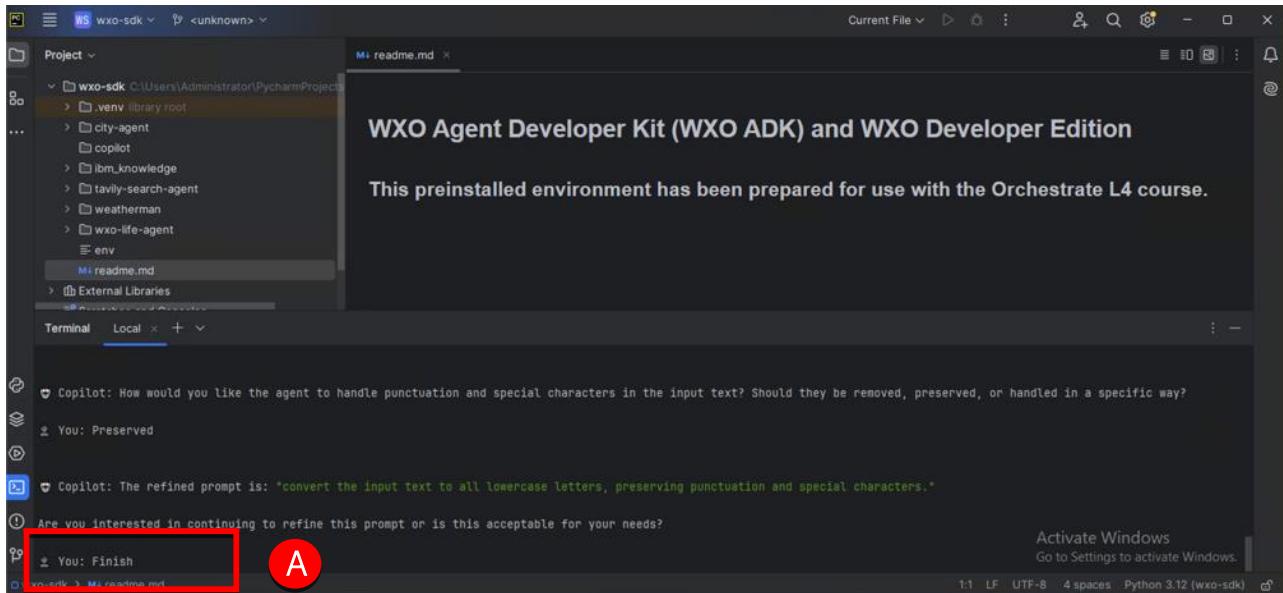
The screenshot shows a continuation of the PyCharm terminal conversation:

```

Copilot: convert the input text to all lowercase letters.
Are you interested in continuing to refine this prompt or is this acceptable for your needs?
You: Yes, let's refine
Copilot: How would you like the agent to handle punctuation and special characters in the input text? Should they be removed, preserved, or handled in a specific way?
You: Preserved
  
```

A red box highlights the user's response "You: Preserved". A red circle with the letter "A" is positioned to the right of the terminal window.

- Finish the copilot session, in our example we have answered 'Finish' (A)



The screenshot shows a PyCharm interface with a terminal window open. The terminal output is as follows:

```

Copilot: How would you like the agent to handle punctuation and special characters in the input text? Should they be removed, preserved, or handled in a specific way?
You: Preserved

Copilot: The refined prompt is: "convert the input text to all lowercase letters, preserving punctuation and special characters."
Are you interested in continuing to refine this prompt or is this acceptable for your needs?
You: Finish

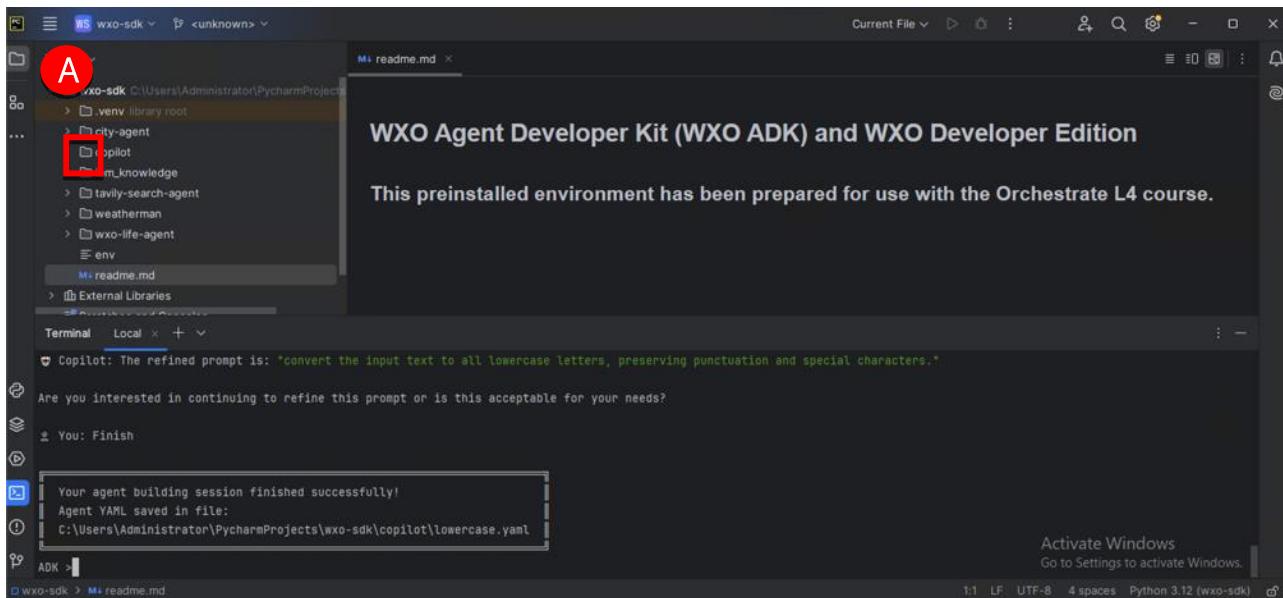
```

A red box highlights the message "You: Finish", which is labeled with a red circle containing the letter "A".

As soon as you stop the refinement, copilot will generate the agent description file as specified in the invocation command.

Review the generated agent

- Expand the copilot folder (A).



The screenshot shows the same PyCharm interface after expanding the "copilot" folder. The terminal output now includes a message box:

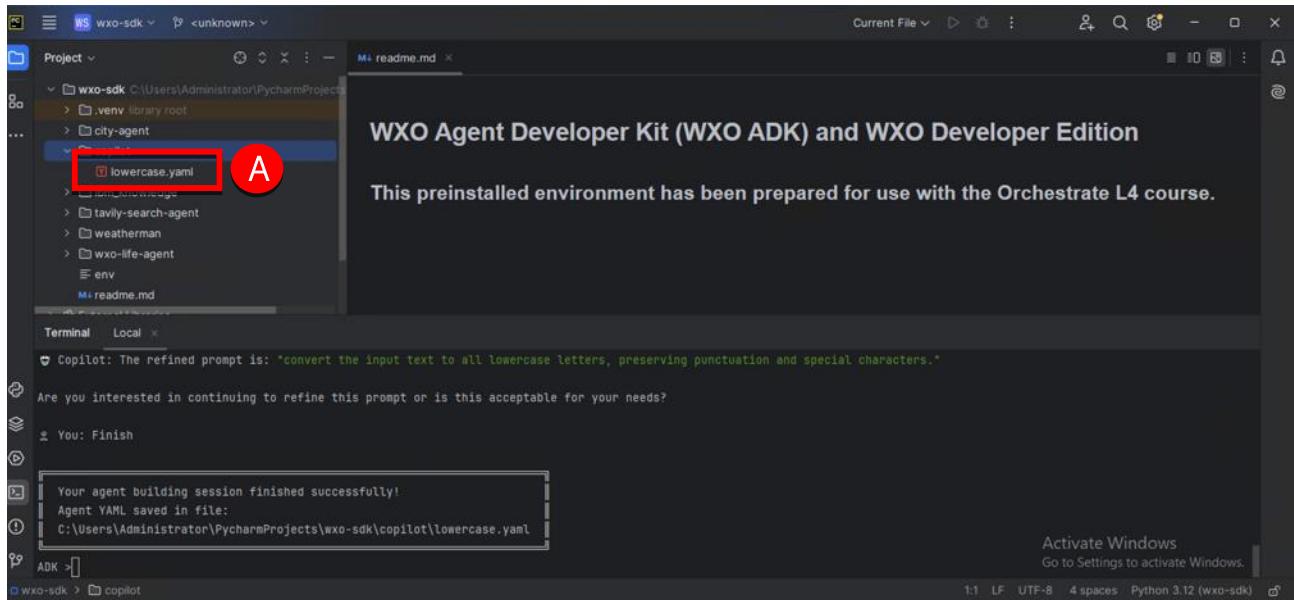
```

Your agent building session finished successfully!
Agent YAML saved in file:
C:\Users\Administrator\PycharmProjects\wxo-sdk\copilot\lowercase.yaml

```

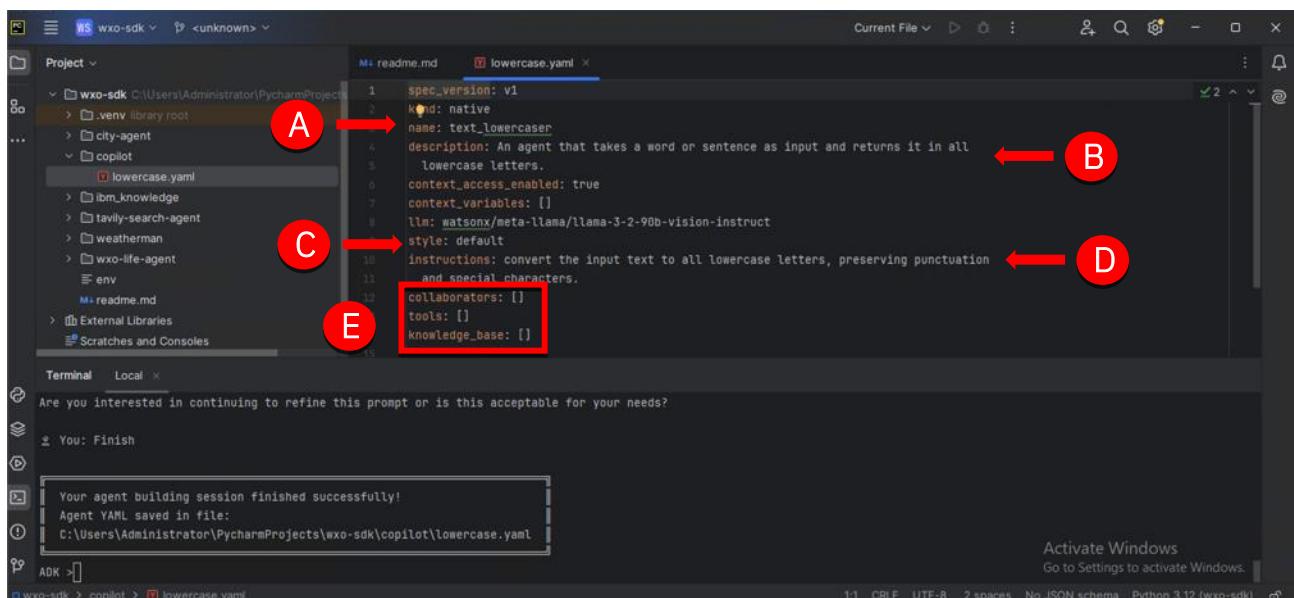
A red circle labeled "A" is positioned over the expanded "copilot" folder in the project tree.

2. Double-click on **lowercase.yaml** (A) to open the file.



3. Review the generated agent description.

- The name copilot gave to your agent (A)
- The agent description generated by copilot (B).
- The agent style 'default' as discussed with copilot (C).
- The instructions generated from you examples and refinement process (D).
- The absence of tools/collaborators and knowledge as specified to copilot (E).



Note: Your own agent will be different based on your conversation with copilot. This agent is provided as an example only.

Note: You can update any of these fields from this editor. You can change the agent's name for example.

Preview your agent

In this section, you will import your lowercase agent to your environment.

1. Enter the following command (A) to import your agent:

```
orchestrate agents import -f ./copilot/lowercase.yaml
```

The screenshot shows the PyCharm IDE interface. On the left, the Project tool window displays a file structure under 'wxo-sdk'. In the center, the code editor shows the 'lowercase.yaml' file with the following content:

```
spec_version: v1
kind: native
name: text_lowecaser
description: An agent that takes a word or sentence as input and returns it in all
    lowercase letters.
context_access_enabled: true
context_variables: []
llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct
style: default
instructions: convert the input text to all lowercase letters, preserving punctuation
    and special characters.
collaborators: []
tools: []
knowledge_base: []
```

Below the code editor is a terminal window. It shows a conversation with a bot:

```
Are you interested in continuing to refine this prompt or is this acceptable for your needs?
>You: Finish
```

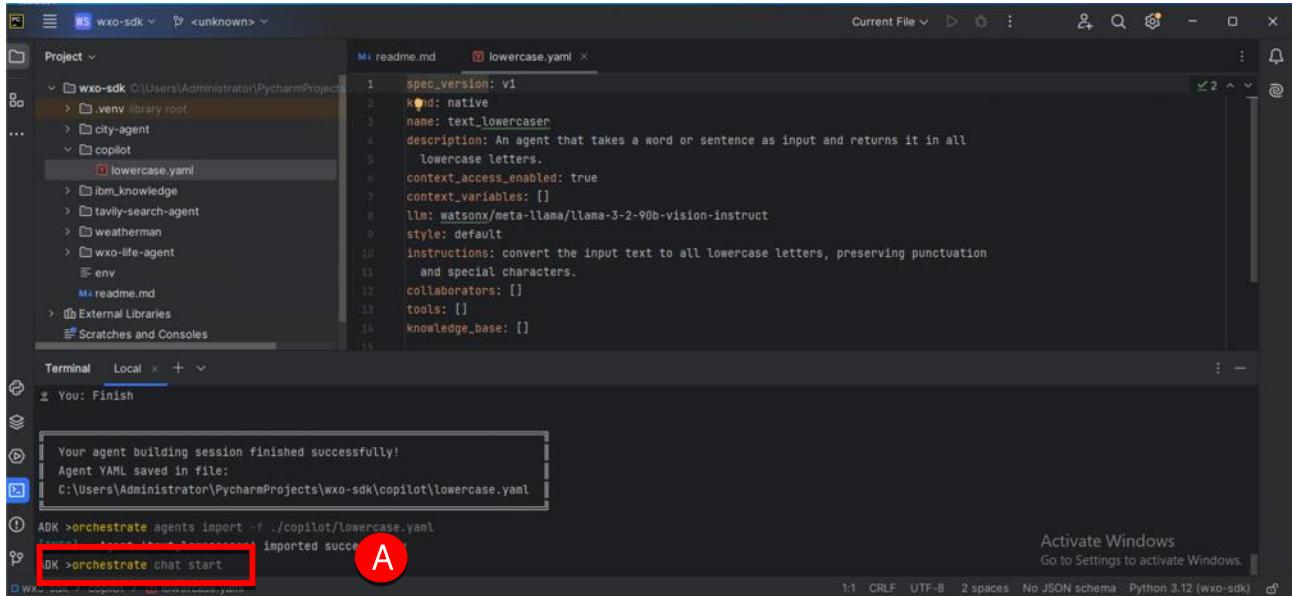
At the bottom of the terminal window, a message indicates the session was successful:

```
Your agent building session finished successfully!
Agent YAML saved in file:
C:\Users\Administrator\PycharmProjects\wxo-sdk\copilot\lowercase.yaml
```

A red box highlights the terminal command 'A' is being typed:

```
A >orchestrate agents import -f ./copilot/lowercase.yaml
```

2. Type 'orchestrate chat start' (A) to start the chat interface.



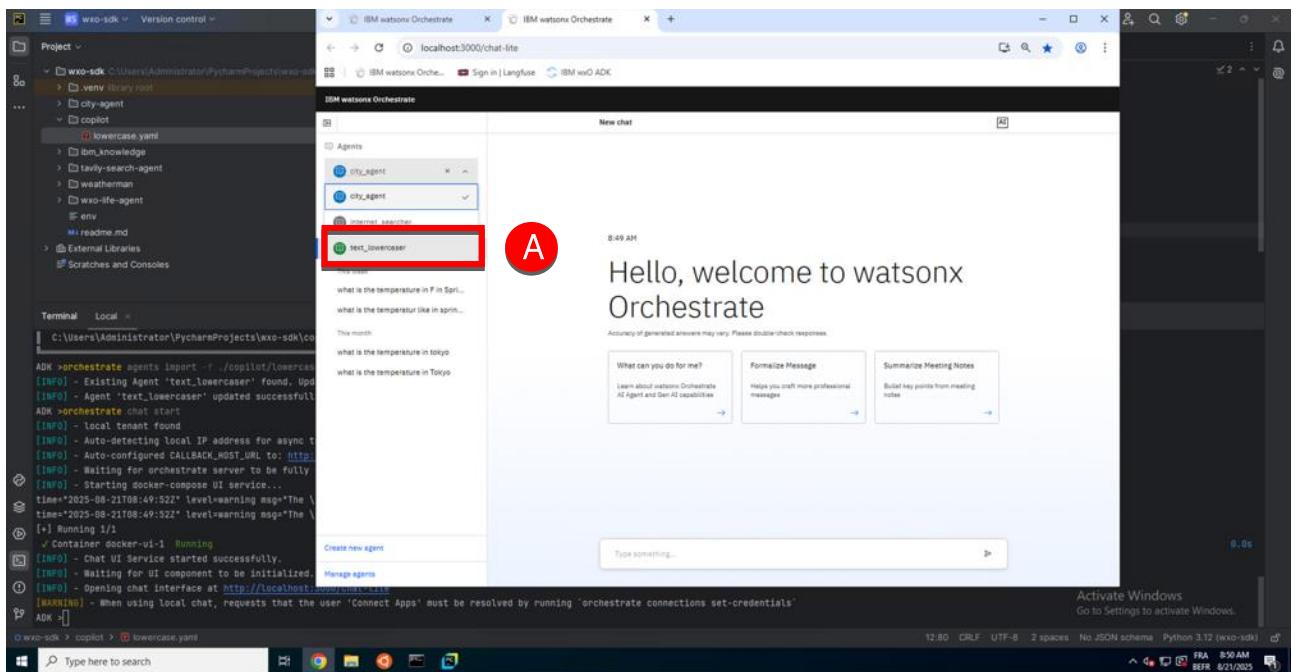
The screenshot shows the PyCharm IDE interface. In the top navigation bar, the project is named 'wxo-sdk'. The left sidebar shows a file tree with a file named 'lowercase.yaml' selected. The main editor window displays the contents of 'lowercase.yaml'. Below the editor is a terminal window. The terminal output shows:

```
Your agent building session finished successfully!
Agent YAML saved in file:
C:\Users\Administrator\PycharmProjects\wxo-sdk\copilot\lowercase.yaml

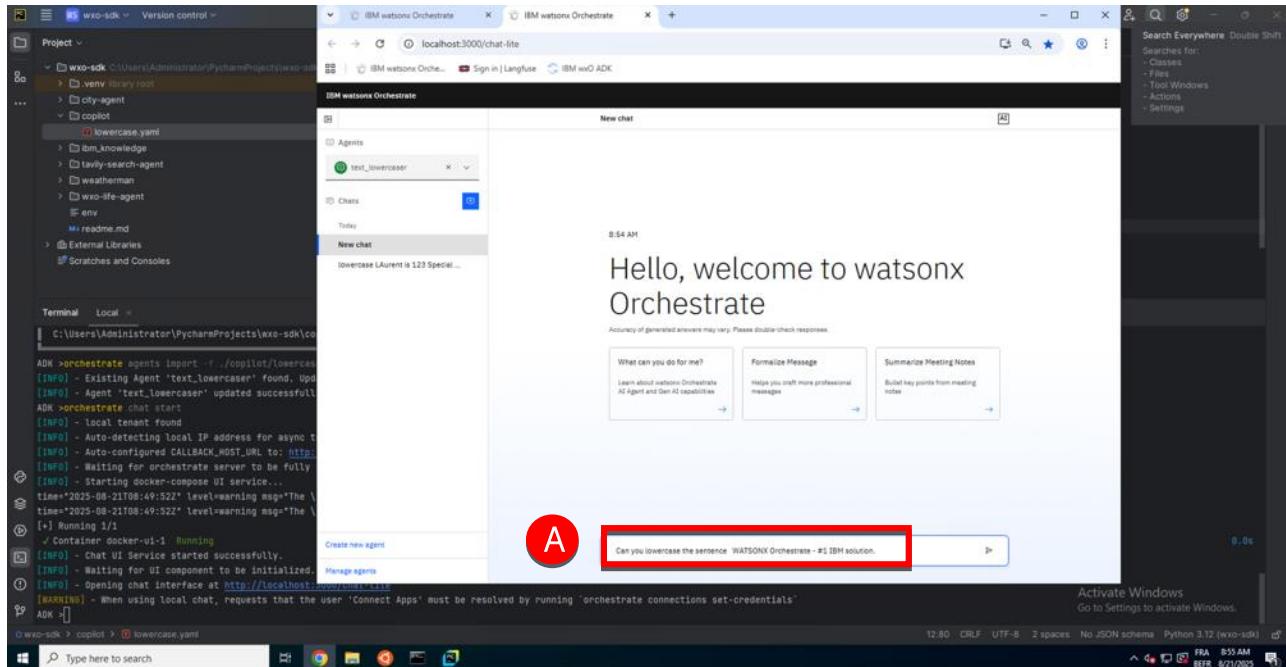
ADK >orchestrate agents import -f ./copilot/lowercase.yaml
ADK >orchestrate chat start imported success
```

A red circle labeled 'A' highlights the command 'ADK >orchestrate chat start' in the terminal.

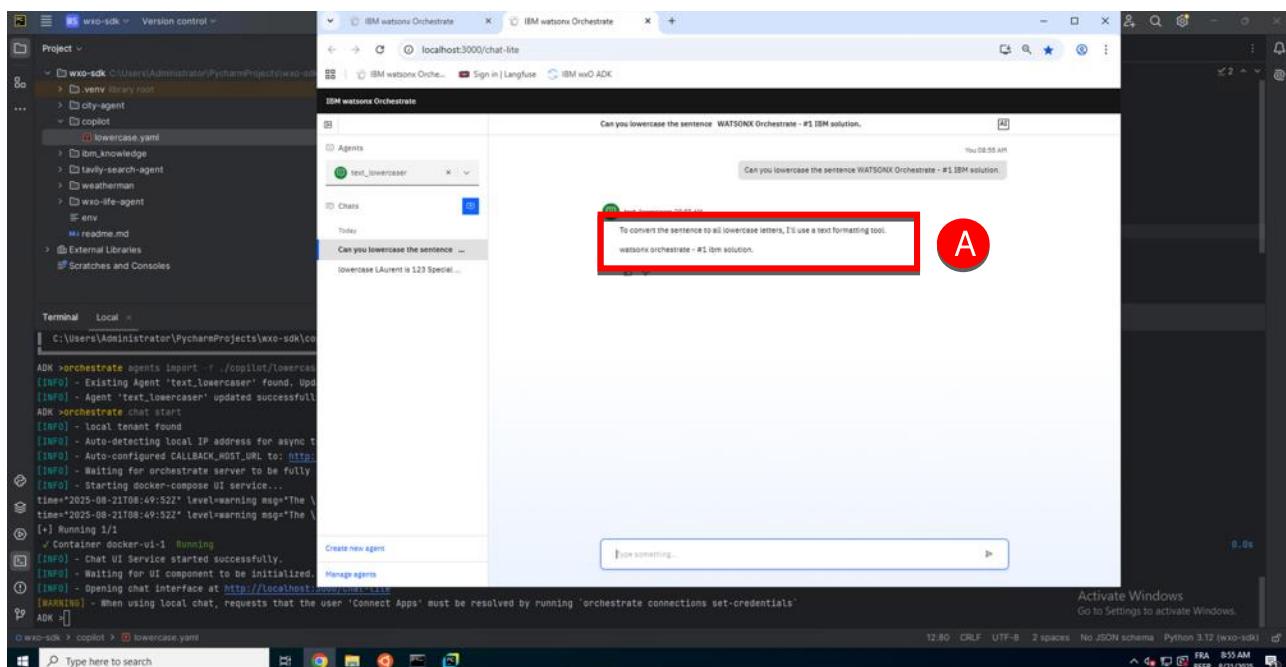
3. Select the **text_lowercase** (use the name specified in your agent yaml file in any case) agent from the agent selector drop-down menu (A).



4. Type 'Can you lowercase the sentence solution.' (A) to test the agent.



5. The agent should return a lowercased sentence (A) as specified during the conversation with copilot.



Note: You can observe that IBM has been lowercased. You could refine this agent not to lowercase IBM or any known agencies/organizations acronyms.

Congratulations, you have generated your first AI agent from scratch using copilot. In the next chapter you will learn how to improve an existing agent using copilot.

Section 2: Refine an existing agent with copilot

In section 1, you have discovered the process to create an agent from scratch using copilot. Copilot is also able to improve an existing agent from its yaml description file. It will follow a similar process from the one you have learned in the previous section.

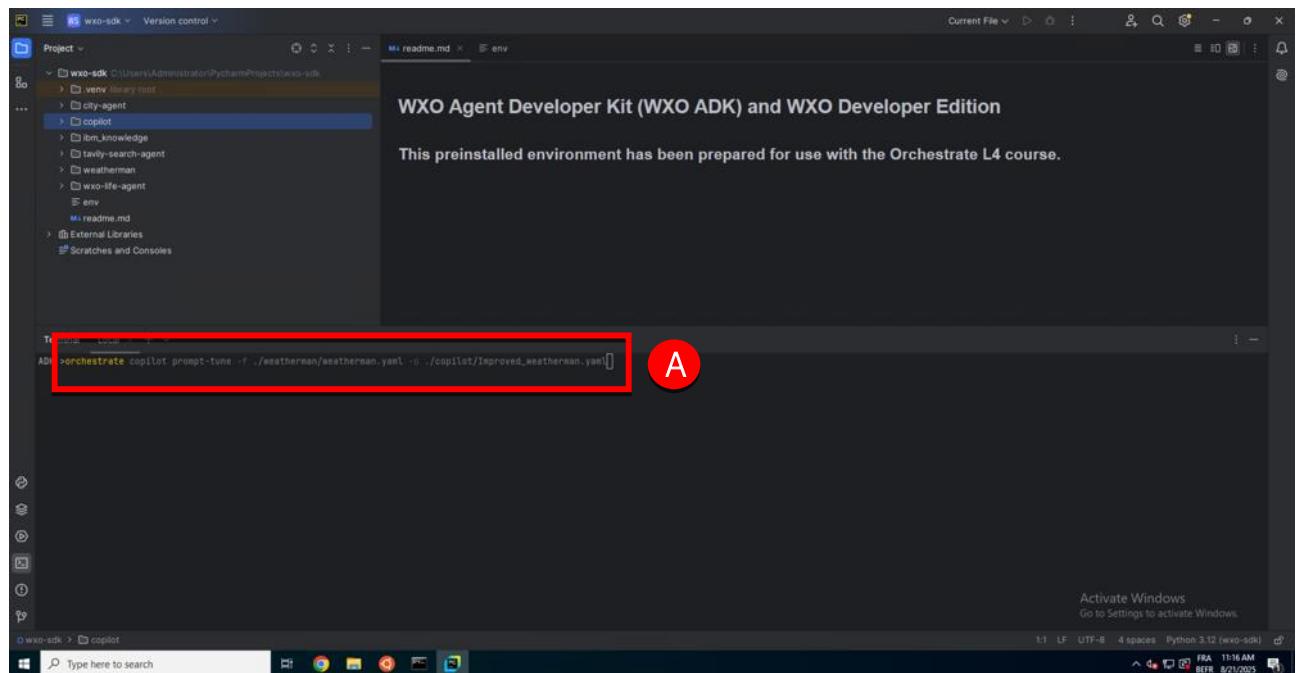
In this section, you will use copilot to improve the weatherman agent you have been using in the ADK-API lab.

During the refining process feel free to answer whatever you want to copilot questions. Stop the refining process when you are OK with the prompt or let copilot continue asking refining questions letting copilot know that you want to continue the refining.

Initiate the refining process

1. Enter the following command to start the refining process (A).

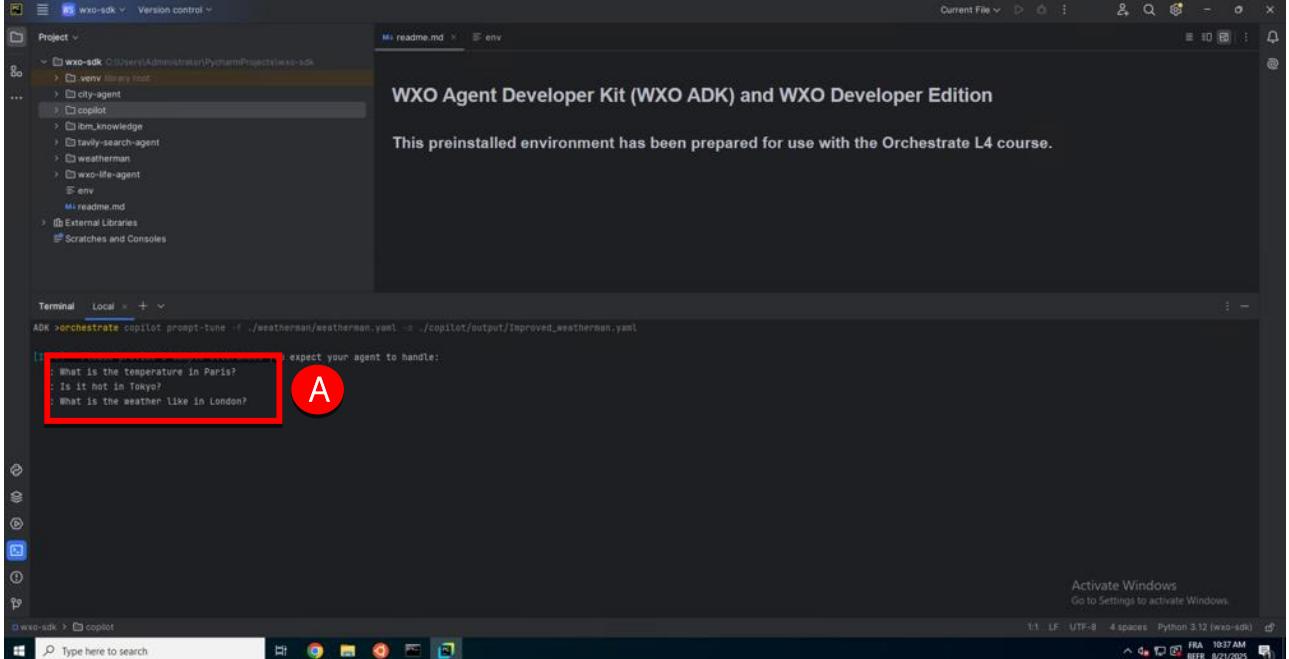
```
orchestrate copilot prompt-tune -f ./weatherman/weatherman.yaml -o  
./copilot/Improved_weatherman.yaml
```



The agent description file is provided as an input (-f file option) and the improved agent description file will be stored in the copilot directory (-o output option). This starts the copilot refine process. Immediately, copilot starts asking for some invocation sentences to improve the prompt.

2. Type the following sentences one-by-one when prompted (A).

What is the temperature in Paris?
 Is it hot in Tokyo?
 What is the weather like in London?



The screenshot shows a PyCharm interface with a terminal window open. The terminal command is:

```
ADK >orchestrator copilot prompt-tune -f ./weatherman/weatherman.yaml -o ./copilot/output/Improved_weatherman.yaml
```

The terminal output shows the following interaction:

```
[INFO] - Please provide 3 sample utterances you expect your agent to handle:  

:: What is the temperature in Paris?  

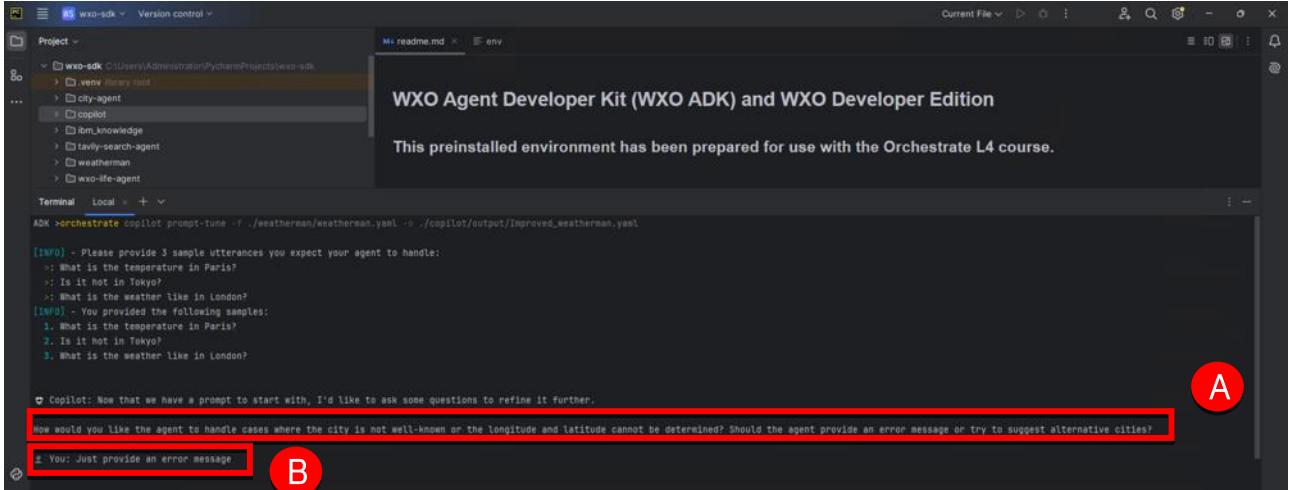
:: Is it hot in Tokyo?  

:: What is the weather like in London?
```

A red box highlights the three sample utterances, and a red circle labeled 'A' is positioned to the right of the terminal window.

Copilot will then start the refining process. It may ask for actions to take when cities are not well known as in this example. In any case, follow your own conversation with the tool.

3. Copilot will come with a first refining question (A). In our case about error management. Answer the question as we did in this example (B).



The screenshot shows the same PyCharm interface with the terminal window. The terminal command and output remain the same as in the previous screenshot.

The terminal output continues with:

```
[INFO] - You provided the following samples:  

1. What is the temperature in Paris?  

2. Is it hot in Tokyo?  

3. What is the weather like in London?
```

Then, Copilot asks:

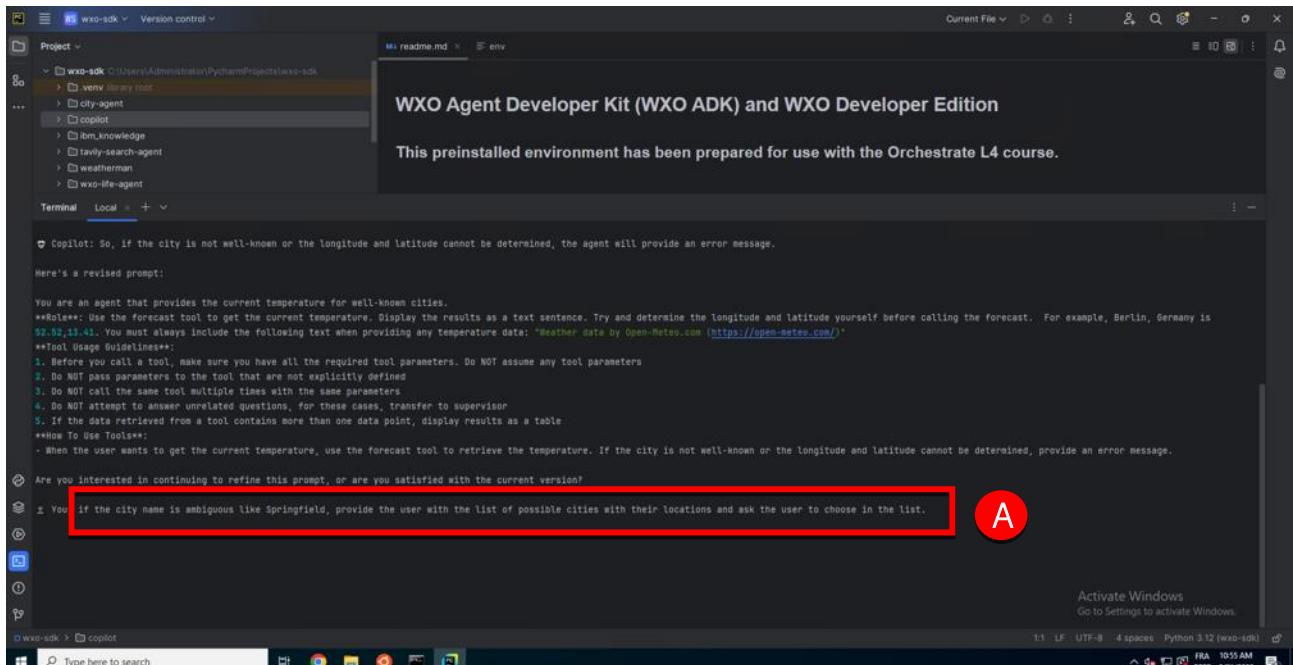
```
Copilot: Now that we have a prompt to start with, I'd like to ask some questions to refine it further.
```

The user's response is highlighted in a red box and labeled 'B' with a red circle:

```
How would you like the agent to handle cases where the city is not well-known or the longitude and latitude cannot be determined? Should the agent provide an error message or try to suggest alternative cities?  

1. You: Just provide an error message
```

4. We will continue with the refinement managing city name ambiguities. It may happen that copilot will have already raised this refinement question. We have typed ‘if the city name is ambiguous like Springfield, provide the user with the list of possible cities with their locations and ask the user to choose in the list.’ (A) as a new refinement.



The screenshot shows a terminal window within a PyCharm project titled 'wxo-sdk'. The terminal displays a refined prompt:

```

Copilot: So, if the city is not well-known or the longitude and latitude cannot be determined, the agent will provide an error message.

Here's a revised prompt:

You are an agent that provides the current temperature for well-known cities.
**Role**: Use the forecast tool to get the current temperature. Display the results as a text sentence. Try and determine the longitude and latitude yourself before calling the forecast. For example, Berlin, Germany is
52.52,13.41. You must always include the following text when providing any temperature data: "Weather data by Open-Meteo.com (https://open-meteo.com/)"

**Tool Usage Guidelines**:
1. Before you call a tool, make sure you have all the required tool parameters. Do NOT assume any tool parameters
2. Do NOT pass parameters to the tool that are not explicitly defined
3. Do NOT call the same tool multiple times with the same parameters
4. Do NOT attempt to answer unrelated questions, for these cases, transfer to supervisor
5. If the data retrieved from a tool contains more than one data point, display results as a table

**How To Use Tools**:
- When the user wants to get the current temperature, use the forecast tool to retrieve the temperature. If the city is not well-known or the longitude and latitude cannot be determined, provide an error message.

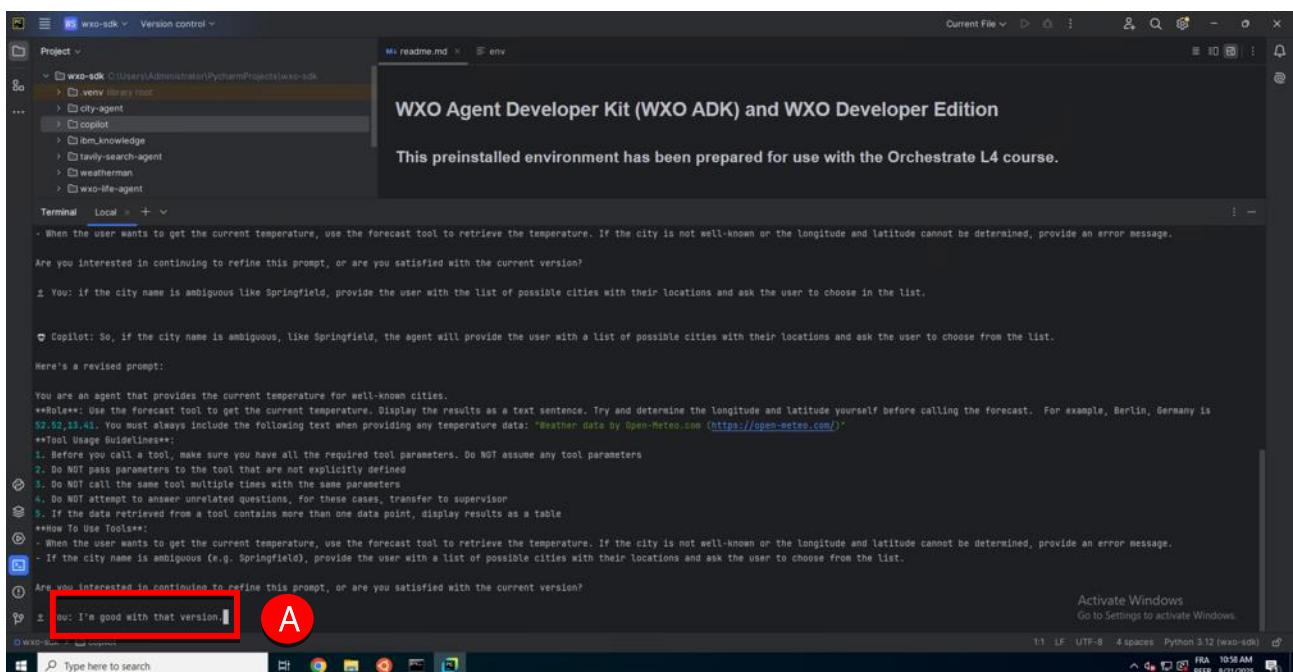
Are you interested in continuing to refine this prompt, or are you satisfied with the current version?

# You: if the city name is ambiguous like Springfield, provide the user with the list of possible cities with their locations and ask the user to choose in the list.

```

A red box highlights the refined prompt, and a red circle with the letter 'A' is positioned to its right.

5. Stop the refining process when you are satisfied with the prompt. Type ‘I’m good with this version’ (A) or just ‘Exit’ to let copilot know you want to stop.



The screenshot shows a terminal window within a PyCharm project titled 'wxo-sdk'. The terminal displays the final refined prompt:

```

Copilot: So, if the city is not well-known or the longitude and latitude cannot be determined, the agent will provide the user with a list of possible cities with their locations and ask the user to choose from the list.

Here's a revised prompt:

You are an agent that provides the current temperature for well-known cities.
**Role**: Use the forecast tool to get the current temperature. Display the results as a text sentence. Try and determine the longitude and latitude yourself before calling the forecast. For example, Berlin, Germany is
52.52,13.41. You must always include the following text when providing any temperature data: "Weather data by Open-Meteo.com (https://open-meteo.com/)"

**Tool Usage Guidelines**:
1. Before you call a tool, make sure you have all the required tool parameters. Do NOT assume any tool parameters
2. Do NOT pass parameters to the tool that are not explicitly defined
3. Do NOT call the same tool multiple times with the same parameters
4. Do NOT attempt to answer unrelated questions, for these cases, transfer to supervisor
5. If the data retrieved from a tool contains more than one data point, display results as a table

**How To Use Tools**:
- When the user wants to get the current temperature, use the forecast tool to retrieve the temperature. If the city is not well-known or the longitude and latitude cannot be determined, provide an error message.
- If the city name is ambiguous (e.g. Springfield), provide the user with a list of possible cities with their locations and ask the user to choose from the list.

Are you interested in continuing to refine this prompt, or are you satisfied with the current version?

# You: I'm good with that version.

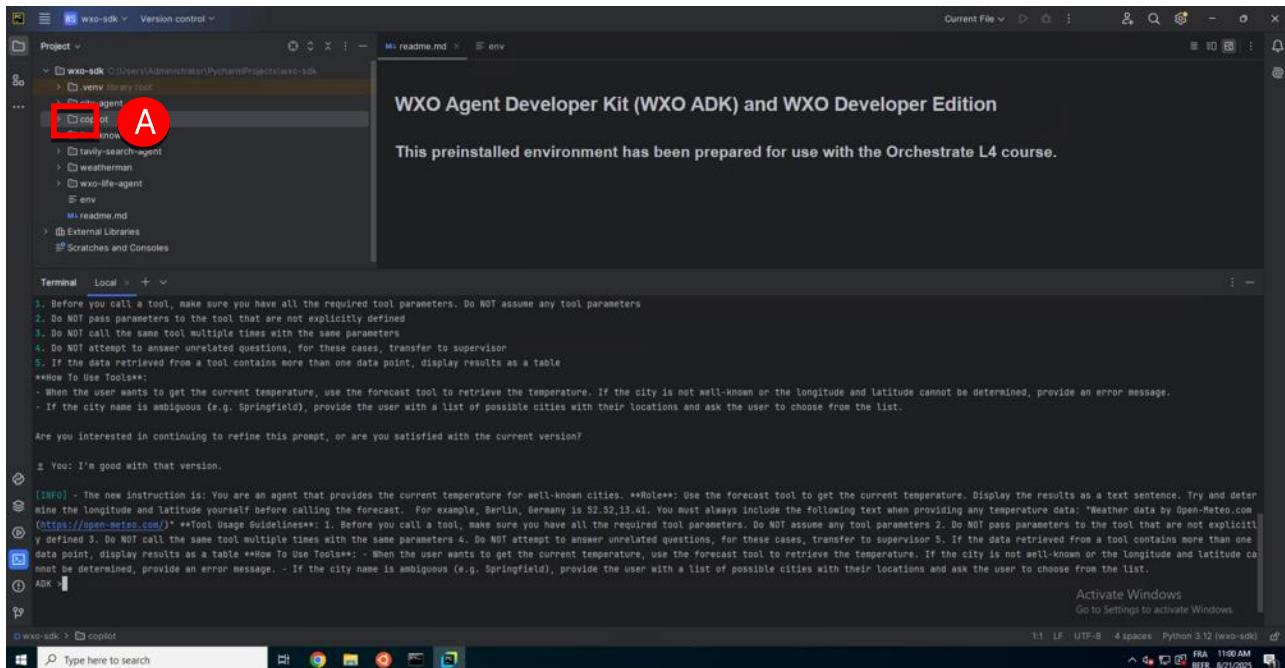
```

A red box highlights the final refined prompt, and a red circle with the letter 'A' is positioned to its right.

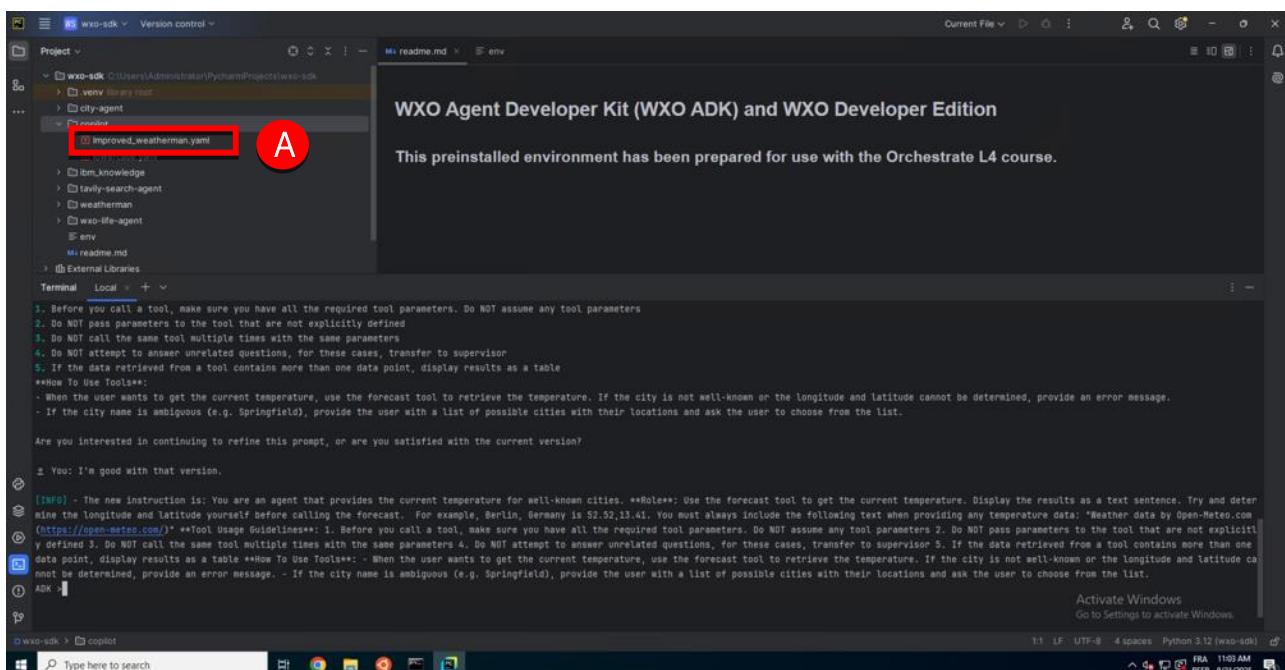
Review the improved agent

You can now review the improved agent description in opening the generated file in your copilot directory.

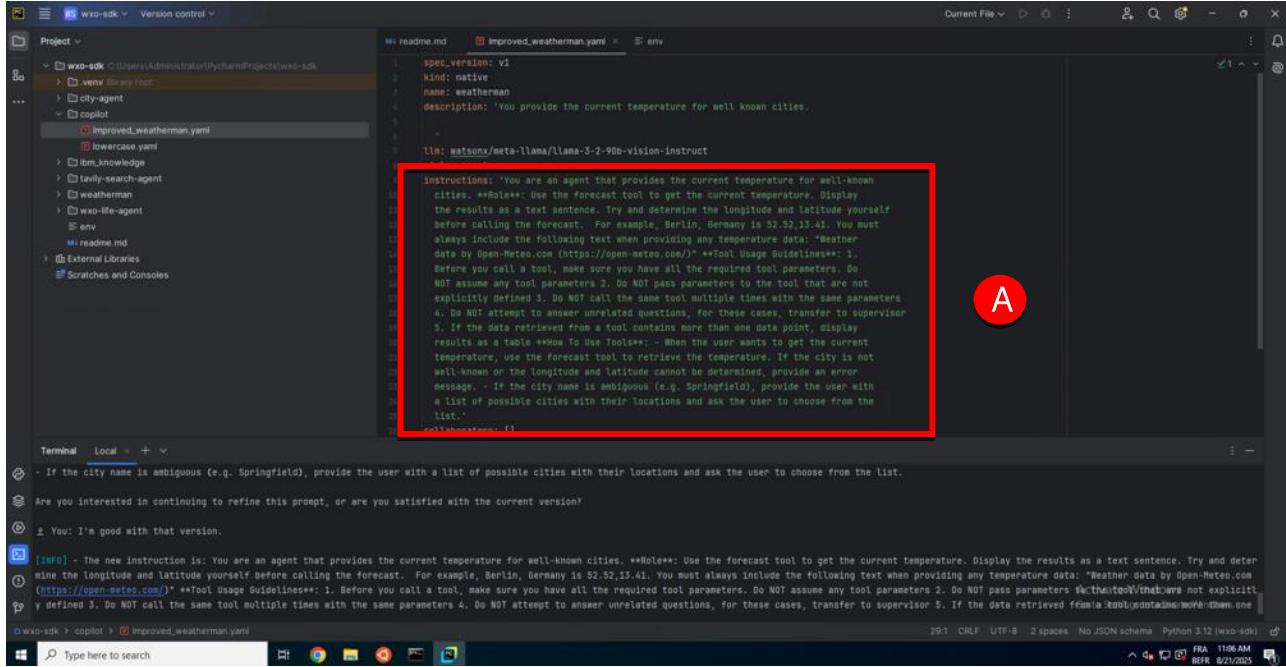
1. Expand the copilot folder (A).



2. Double click the Improved_weatherman.yaml file (A) to open it.



3. Read the new instructions (A) that have been improved by copilot. Note that the initial constraints to include the weather forecast source for example are still in there.



```

spec_version: v1
kind: native
name: weatherman
description: 'You provide the current temperature for well known cities.'

llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct

instructions: 'You are an agent that provides the current temperature for well-known cities. **Role**: Use the forecast tool to get the current temperature. Display the results as a text sentence. Try and determine the longitude and latitude yourself before calling the forecast. For example, Berlin, Germany is 52.52,13.41. You must always include the following text when providing any temperature data: "Weather data by Open-Meteo.com (https://open-meteo.com/)" **Tool Usage Guidelines**: 1. Before you call a tool, make sure you have all the required tool parameters. Do NOT assume any tool parameters 2. Do NOT pass parameters to the tool that are not explicitly defined 3. Do NOT call the same tool multiple times with the same parameters 4. Do NOT attempt to answer unrelated questions, for these cases, transfer to supervisor 5. If the data retrieved from a tool contains more than one data point, display results as a table **How To Use Tools** - When the user wants to get the current temperature, use the forecast tool to retrieve the temperature. If the city is not well-known or the longitude and latitude cannot be determined, provide an error message. - If the city name is ambiguous (e.g. Springfield), provide the user with a list of possible cities with their locations and ask the user to choose from the list.'

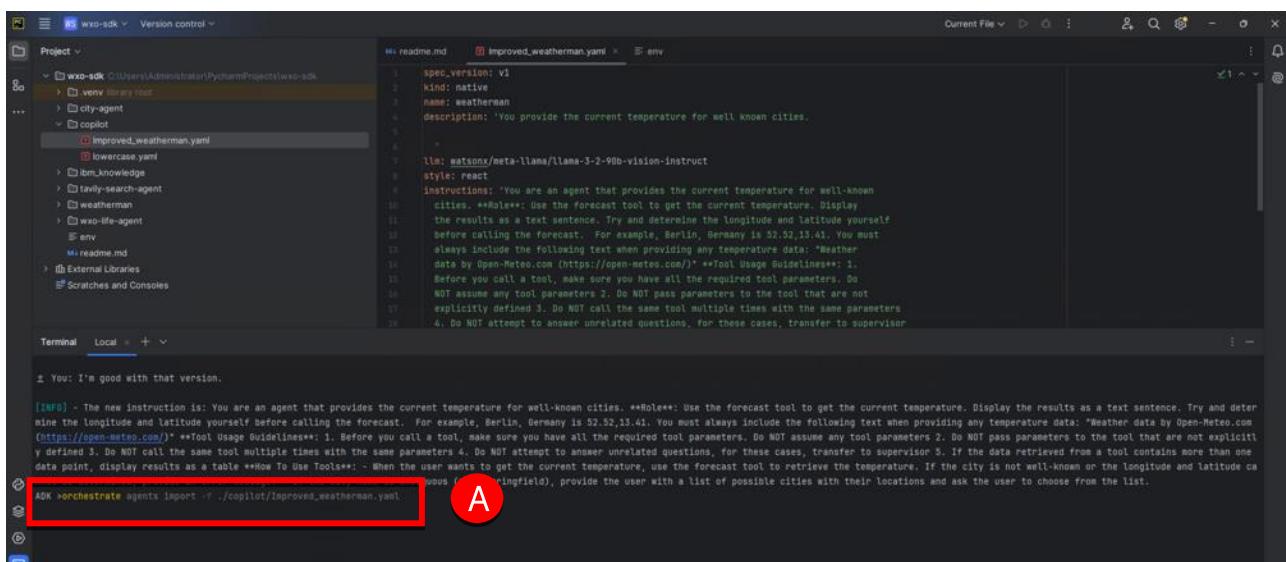
```

Preview the agent

In this section, you will import your lowercase agent to your environment.

- Enter the following command (A) to import your agent:

```
orchestrate agents import -f ./copilot/Improved_weatherman.yaml
```

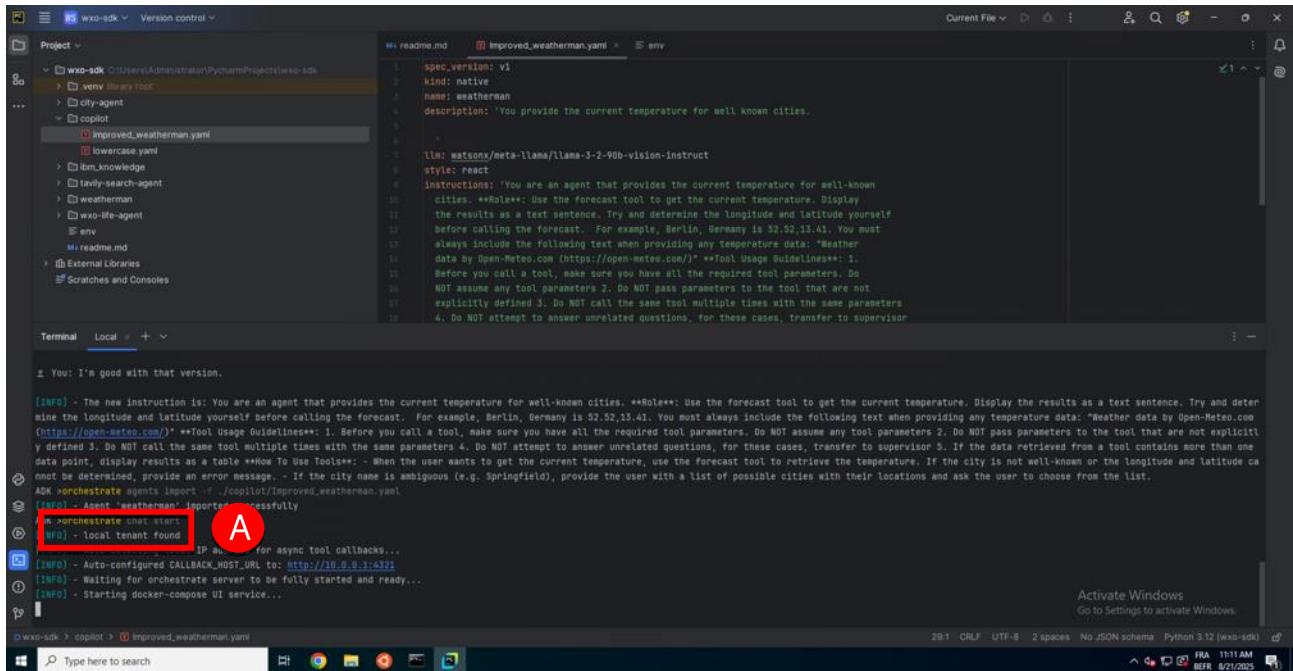


```

$ orchestrate agents import -f ./copilot/Improved_weatherman.yaml

```

2. Type 'orchestrate chat start' (A) to start the chat interface.

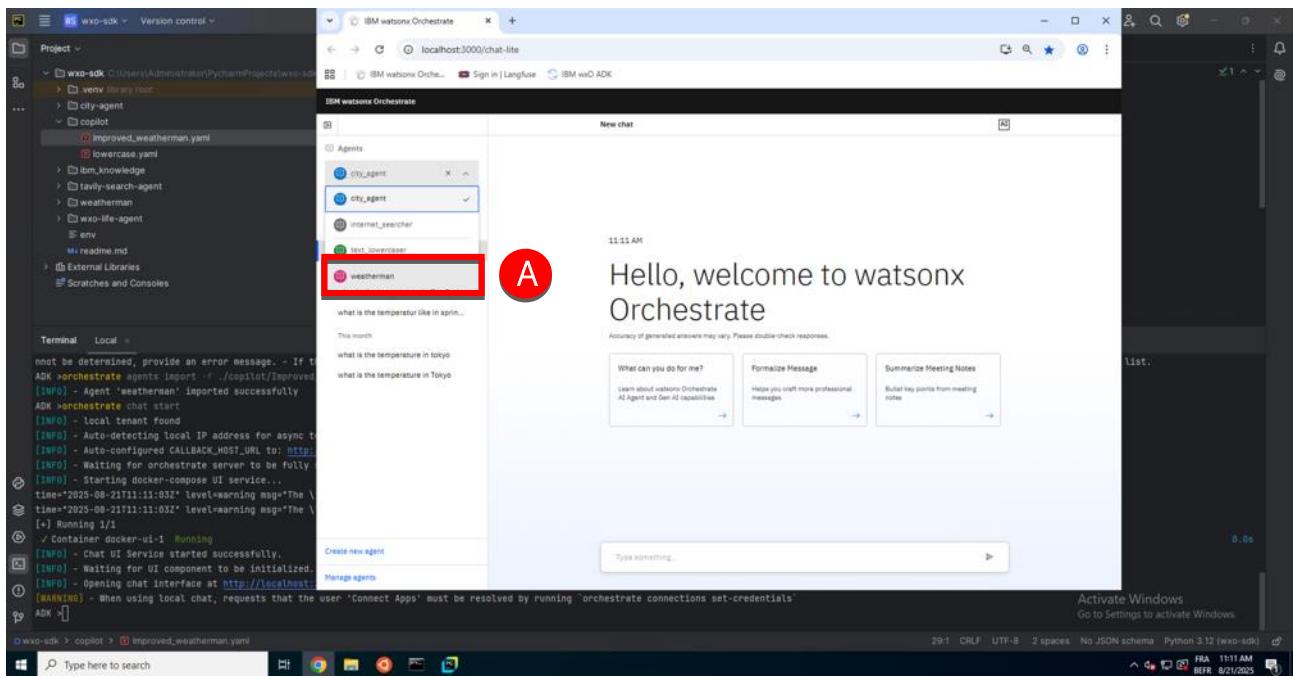


```

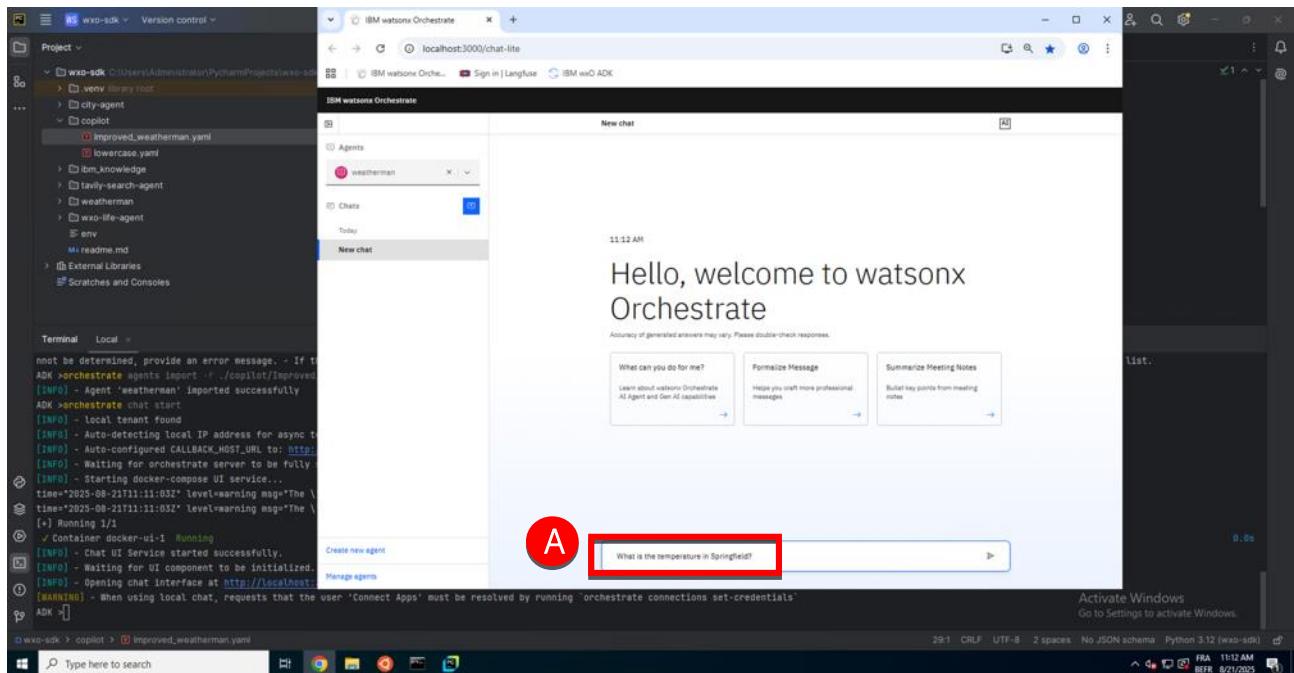
[INFO] - The new instruction is: You are an agent that provides the current temperature for well-known cities. **Role**: Use the forecast tool to get the current temperature. Display the results as a text sentence. Try and determine the longitude and latitude yourself before calling the forecast. For example, Berlin, Germany is 52.52,13.41. You must always include the following text when providing any temperature data: "Weather data by Open-Meteo (https://open-meteo.com/)" **Tool Usage Guidelines**: 1. Before you call a tool, make sure you have all the required tool parameters. Do NOT assume any tool parameters 2. Do NOT pass parameters to the tool that are not explicitly defined 3. Do NOT call the same tool multiple times with the same parameters 4. Do NOT attempt to answer unrelated questions. For these cases, transfer to supervisor
[INFO] - Agent 'weatherman' imported successfully
[INFO] - Local tenant found
[INFO] - IP address for sync tool callbacks...
[INFO] - Auto-configured CALLBACK_HOST_URL to: http://10.0.0.1:4321
[INFO] - Waiting for orchestrate server to be fully started and ready...
[INFO] - Starting docker-compose UI service...

```

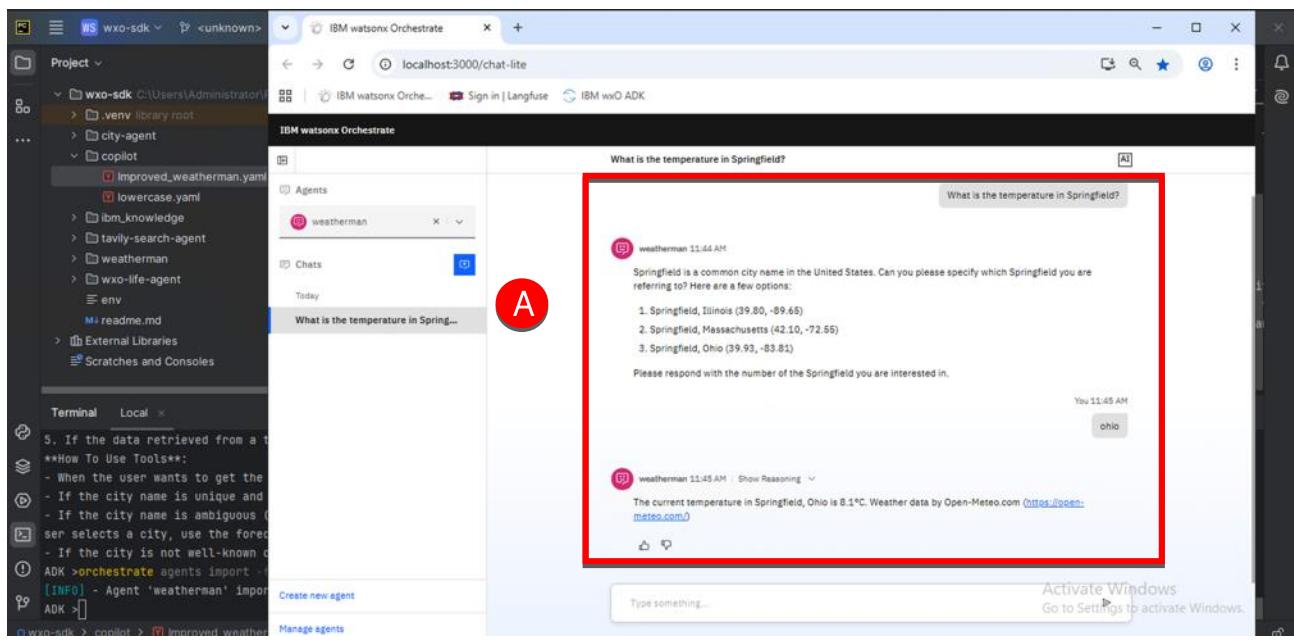
3. Select the **weatherman** agent from the agent selector drop-down menu (A).



4. Type 'What is the temperature in Springfield?' (A) to test the agent.



5. The agent should display a list of different cities (A) as specified during the conversation with copilot.



Congratulations, you have learned how to use copilot to improve an existing agent.

Section 3: Create a complex Agentic AI agent with copilot

Now that you are familiar with copilot, you can use it for more complex agent creation. In this section you will use re-create the 3 agents from the ‘Collaborating agents with Python’ lab. The required tools are already imported in your environment. Copilot will see them and suggest using them when it seems appropriate.

Part 1: Create a Life insurance agent

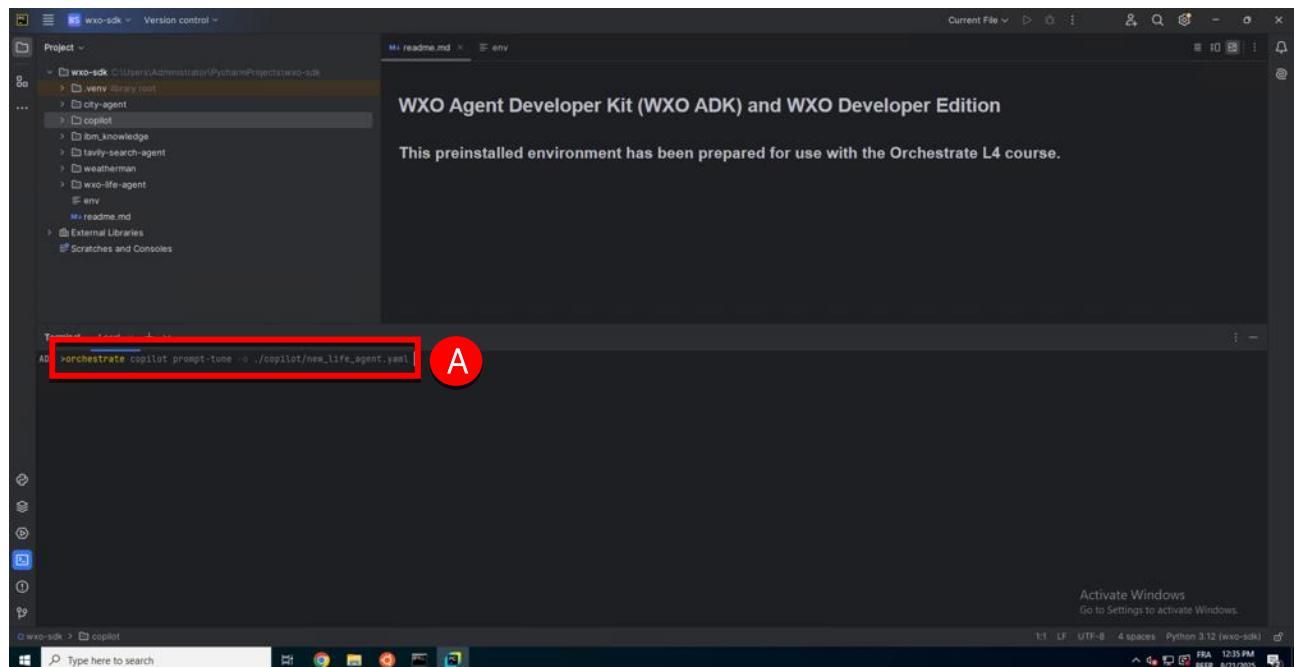
In this first part, you will use copilot to create the Life insurance agent. You will drive the conversation with copilot so that it uses the ‘life quote’ tool.

This conversation is an example of a thread to achieve the creation of the life agent.

Start copilot to create the agent

1. Type the following command to start copilot to create the new life agent from scratch.

```
orchestrate copilot prompt-tune -o ./copilot/new_life_agent.yaml
```



2. Type 'Provide a life insurance illustration that includes the annual premium of the policy.' (A).

```
ADK >orchestrator copilot prompt-tune -o ./copilot/new_life_agent.yaml

Copilot: Let's start creating a configuration for your WKO agent.
To begin, could you please describe the desired functionality of your agent? What task do you want it to perform?
you: Provide a life insurance illustration that includes the annual premium of the policy.
```

3. Next copilot offers to select one of the available tools. Let's specify 'use life_quote' (A).

```
You: Provide a life insurance illustration that includes the annual premium of the policy.

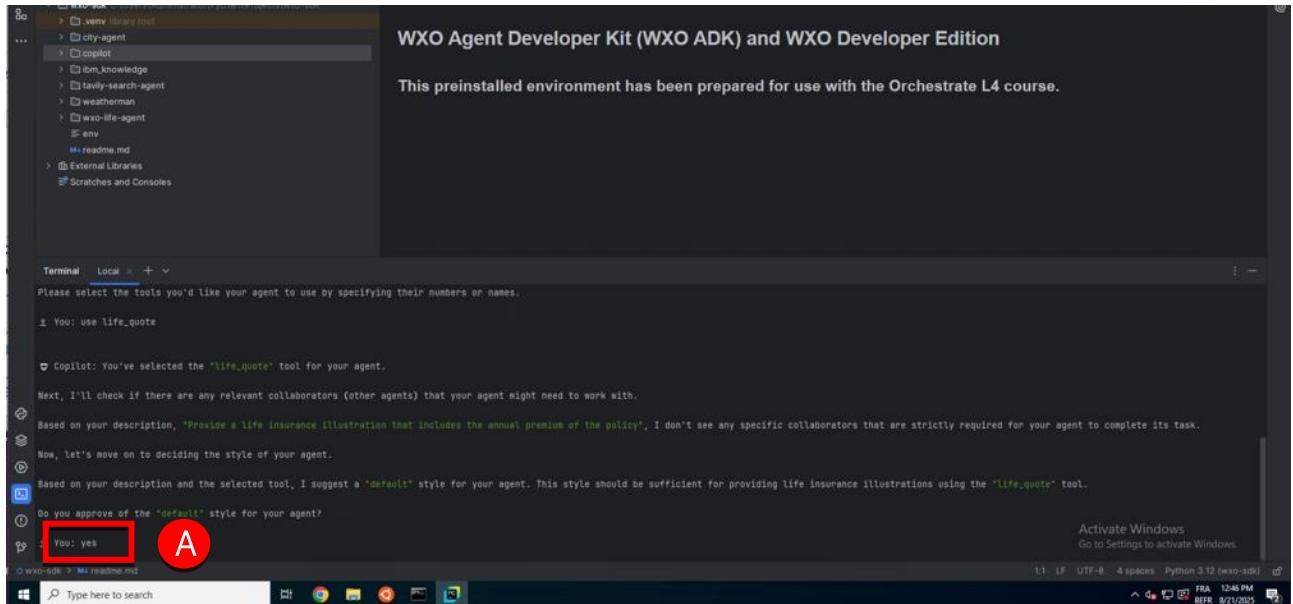
Copilot: Based on your description, "Provide a life insurance illustration that includes the annual premium of the policy", I recommend the following tools for your agent:
1. interest_rate_lookup
2. life_quote
3. monthly_payment_calc

These tools seem relevant to calculating and providing life insurance illustrations, including annual premiums.

The full list of available tools has 5 items. Would you like to see the full list?

Please select the tools you'd like your agent to use by specifying their numbers or names.
you: use life_quote
```

4. Next Copilot will ask about the style of agent. Copilot suggests the default type that is an appropriate choice. We have replied 'yes' to copilot confirmation request (A).



```

WXO Agent Developer Kit (WXO ADK) and WXO Developer Edition
This preinstalled environment has been prepared for use with the Orchestrate L4 course.

Please select the tools you'd like your agent to use by specifying their numbers or names.

# You: use life_quote

Copilot: You've selected the "life_quote" tool for your agent.

Next, I'll check if there are any relevant collaborators (other agents) that your agent might need to work with.

Based on your description, "Provide a life insurance illustration that includes the annual premium of the policy", I don't see any specific collaborators that are strictly required for your agent to complete its task.

Now, let's move on to deciding the style of your agent.

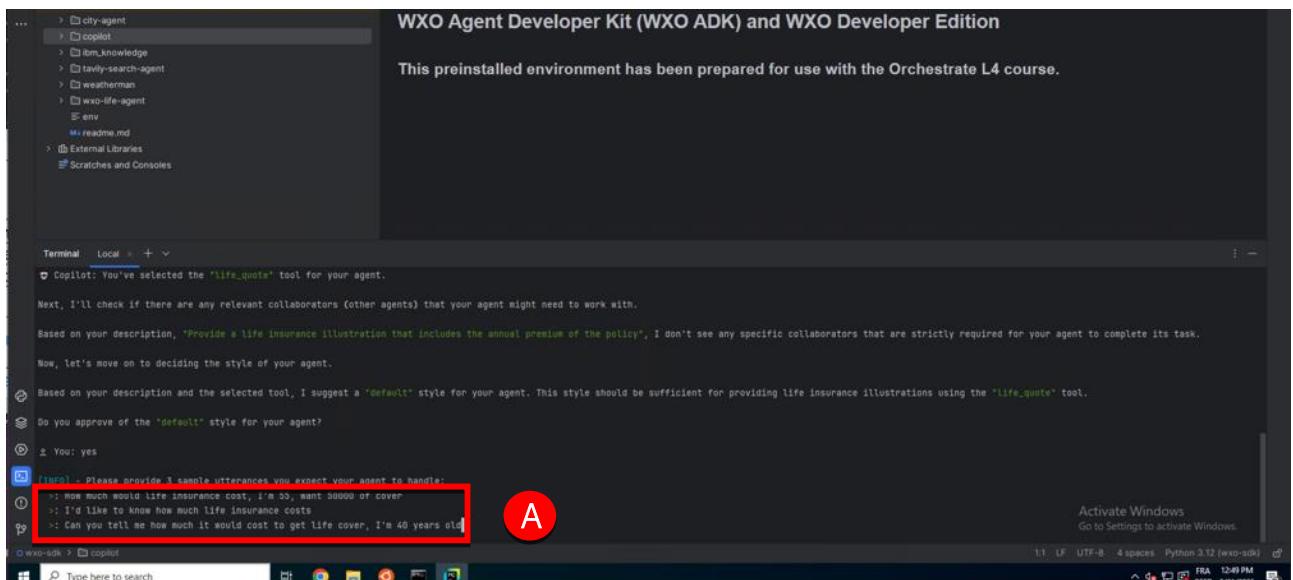
Based on your description and the selected tool, I suggest a "default" style for your agent. This style should be sufficient for providing life insurance illustrations using the "life_quote" tool.

Do you approve of the "default" style for your agent?

# You: yes
  
```

5. Next copilot asks for the 3 invocation sentences. You can enter the 3 following sentences on-by-one (A).

How much would life insurance cost, I'm 55, want 50000 of cover
 I'd like to know how much life insurance costs
 Can you tell me how much it would cost to get life cover; I'm 40 years old



```

WXO Agent Developer Kit (WXO ADK) and WXO Developer Edition
This preinstalled environment has been prepared for use with the Orchestrate L4 course.

Copilot: You've selected the "life_quote" tool for your agent.

Next, I'll check if there are any relevant collaborators (other agents) that your agent might need to work with.

Based on your description, "Provide a life insurance illustration that includes the annual premium of the policy", I don't see any specific collaborators that are strictly required for your agent to complete its task.

Now, let's move on to deciding the style of your agent.

Based on your description and the selected tool, I suggest a "default" style for your agent. This style should be sufficient for providing life insurance illustrations using the "life_quote" tool.

Do you approve of the "default" style for your agent?

# You: yes
  
```

[User]: Please provide 3 sample utterances you expect your agent to handle:

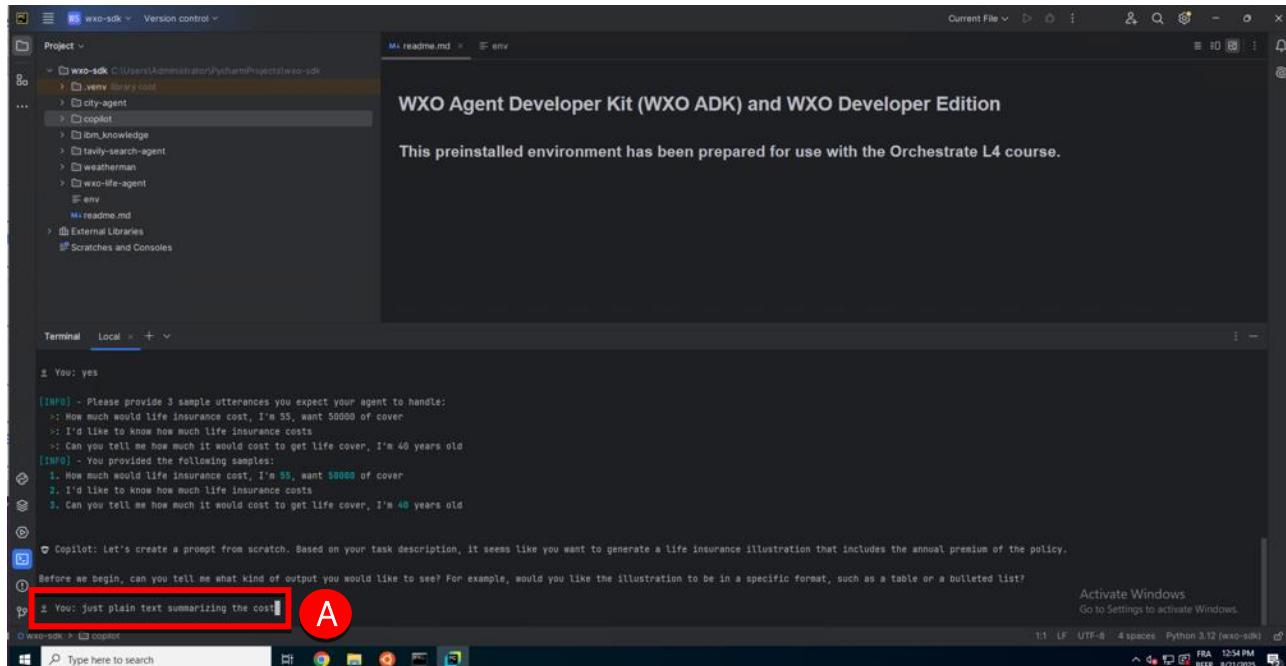
:> How much would life insurance cost, I'm 55, want 50000 of cover
 :> I'd like to know how much life insurance costs
 :> Can you tell me how much it would cost to get life cover, I'm 40 years old

Then the refining process starts.

Refine the prompt

Here again the refine sequence is provided as an example. You will follow your own conversation with copilot trying to stick to the recommendation made in this example.

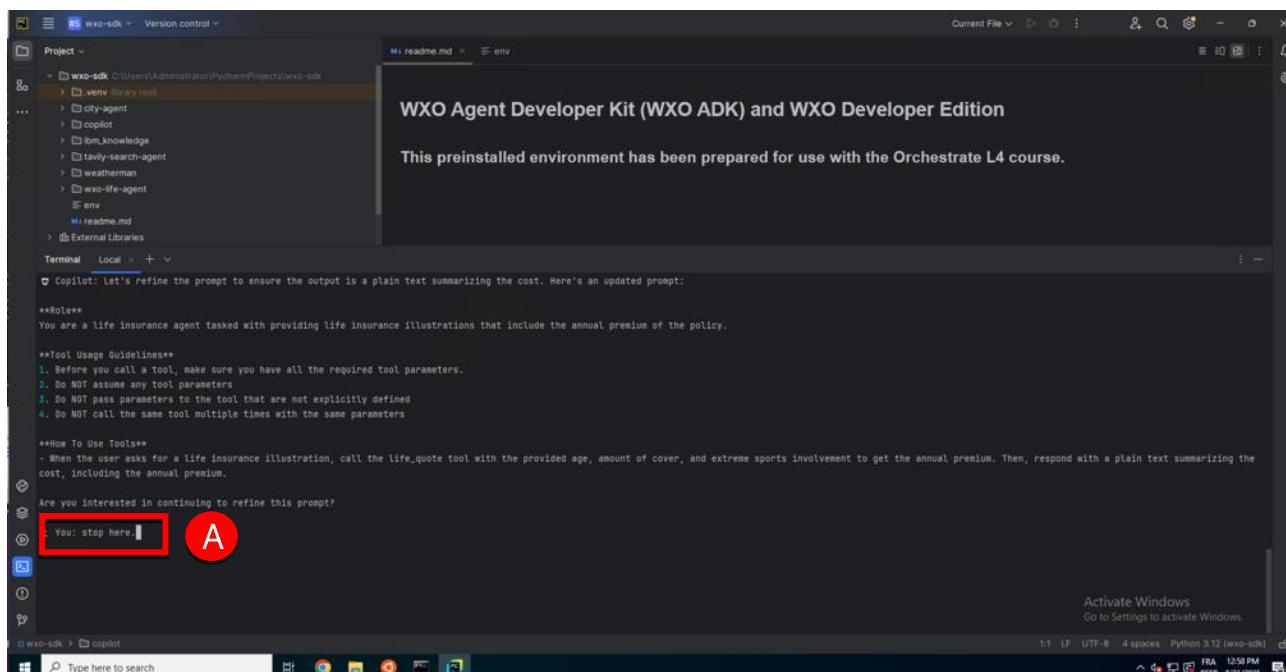
1. In our example, copilots ask about the output format. We enter ‘just plain text summarizing the cost’ (A).



The screenshot shows a PyCharm IDE interface. On the left is a project tree for 'wxo-sdk'. The main window displays a terminal session. The user has typed 'You: just plain text summarizing the cost' into the terminal, which is highlighted with a red box and labeled 'A'. The copilot's response follows:

```
[INFO] - Please provide 3 sample utterances you expect your agent to handle:  
> : How much would life insurance cost, I'm 55, want 50000 of cover  
> : I'd like to know how much life insurance costs  
> : Can you tell me how much it would cost to get life cover, I'm 40 years old  
[INFO] - You provided the following samples:  
1. How much would life insurance cost, I'm 55, want 50000 of cover  
2. I'd like to know how much life insurance costs  
3. Can you tell me how much it would cost to get life cover, I'm 40 years old  
  
Copilot: Let's create a prompt from scratch. Based on your task description, it seems like you want to generate a life insurance illustration that includes the annual premium of the policy.  
Before we begin, can you tell me what kind of output you would like to see? For example, would you like the illustration to be in a specific format, such as a table or a bulleted list?  
You: just plain text summarizing the cost A
```

2. Let copilot know when you are satisfied with the refinement (A).

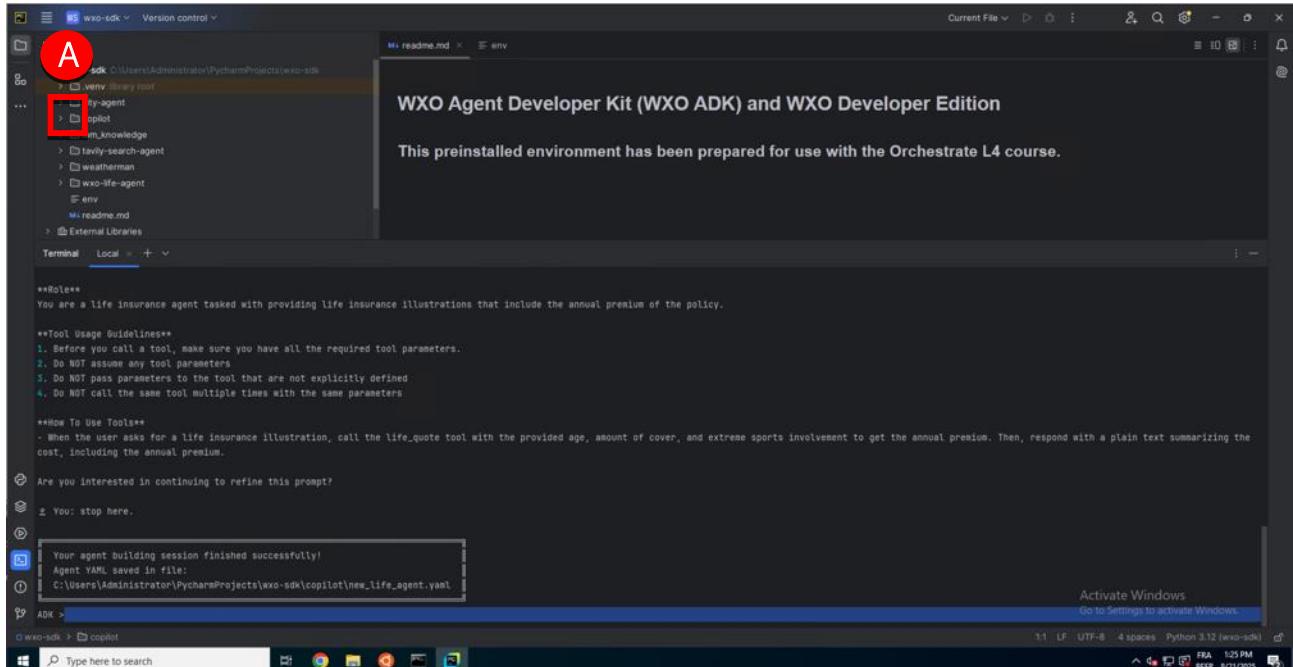


The screenshot shows the same PyCharm IDE interface. The terminal window now contains the refined prompt and the copilot's response:

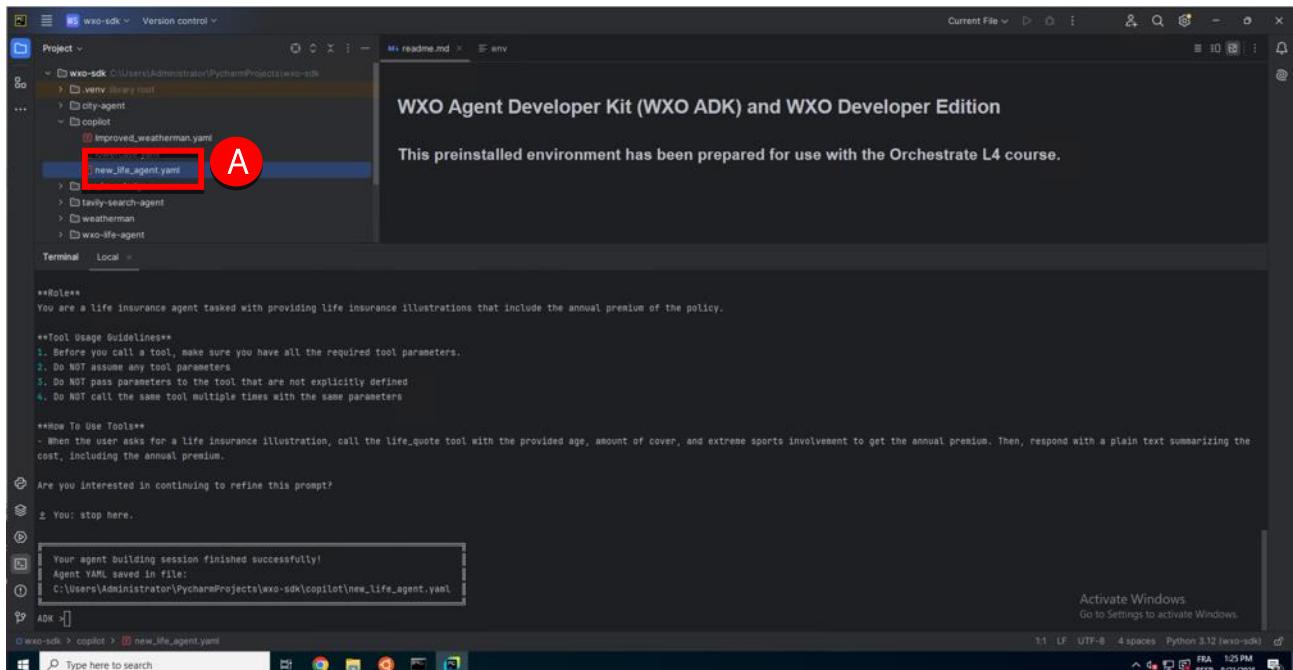
```
copilot: Let's refine the prompt to ensure the output is a plain text summarizing the cost. Here's an updated prompt:  
  
++Role++  
You are a life insurance agent tasked with providing life insurance illustrations that include the annual premium of the policy.  
  
++Tool Usage Guidelines++  
1. Before you call a tool, make sure you have all the required tool parameters.  
2. DO NOT assume any tool parameters.  
3. DO NOT pass parameters to the tool that are not explicitly defined  
4. DO NOT call the same tool multiple times with the same parameters  
  
++How To Use Tools++  
- When the user asks for a life insurance illustration, call the life_quote tool with the provided age, amount of cover, and extreme sports involvement to get the annual premium. Then, respond with a plain text summarizing the cost, including the annual premium.  
  
Are you interested in continuing to refine this prompt?  
You: stop here A
```

Review the agent

1. Expand the copilot folder (A).



2. Double-click on **new_life_agent.yaml** (A) to open the file.



3. Review the generated agent description.

- The name copilot gave to your agent (A)
- The agent description generated by copilot (B).
- The agent style ‘default’ as discussed with copilot (C).
- The instructions generated from you examples and refinement process (D).
- The use of the life_quote tool as specified to copilot (E).

The screenshot shows a PyCharm interface with a project named 'wxo-sdk'. The 'new_life_agent.yaml' file is open in the editor. The code defines an agent with the following specifications:

```

spec_version: v1
kind: native
name: life_insurance_illustrator
description: Provide a life insurance illustration that includes the annual premium of the policy.
context_access_enabled: true
context_variables: []
llm: watsonx/meta-llama/llma-3-2-90b-vision-instruct
style: default

**Tool Instructions:**
++Roles++
You are a life insurance agent tasked with providing life insurance illustrations that include the annual premium of the policy.

**Tool Usage Guidelines**
1. Before you call a tool, make sure you have all the required tool parameters.
2. Do NOT assume any tool parameters
3. Do NOT pass parameters to the tool that are not explicitly defined
4. Do NOT call the same tool multiple times with the same parameters

**How To Use Tools**
- When the user asks for a life insurance illustration, call the life_quote tool with the provided age, amount of cover, and extreme sports involvement to generate the illustration.

tools:
- life_quote
knowledge_base: []

```

Red circles labeled A through E point to the following areas:

- A:** Points to the 'copilot' folder in the project tree.
- B:** Points to the 'description' field in the YAML code.
- C:** Points to the 'style: default' line in the YAML code.
- D:** Points to the 'How To Use Tools' section in the terminal output.
- E:** Points to the 'life_quote' tool entry in the YAML code.

The terminal window shows the completion of the agent building session and the saving of the Agent YAML file.

Part 2: Create a Monthly payment agent

In this second part, like part 1, you will use copilot to create the Monthly payment agent.

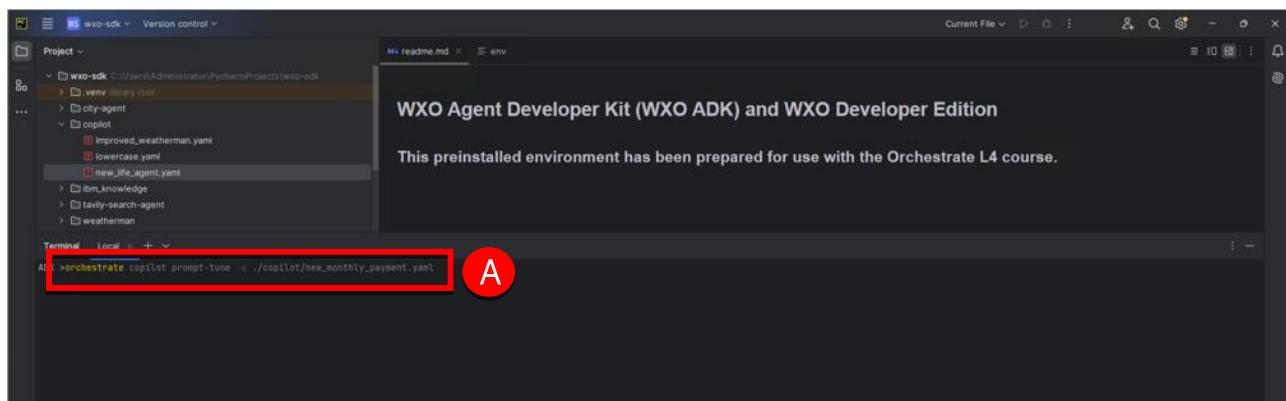
This agent will use 2 tools:

- interest_rate_lookup
- monthly_payment_calc

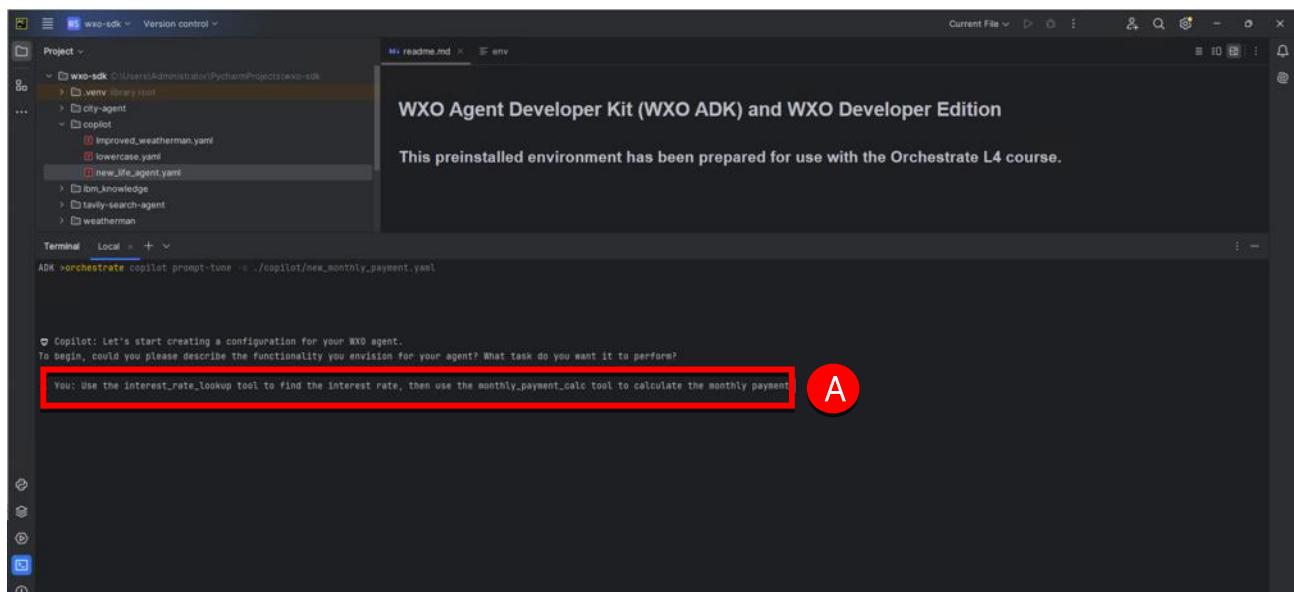
Start copilot to create the agent

1. Type the following command to start copilot to create the new life agent from scratch.

```
orchestrate copilot prompt-tune -o ./copilot/new_monthly_payment.yaml
```



2. Type 'Use the interest_rate_lookup tool to find the interest rate, then use the monthly_payment_calc tool to calculate the monthly payment' (A).



3. Next, copilot asks for the tools to use. We will use the 2 ones suggested by copilot (A).

WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

```
ADK >orchestrate copilot prompt-tune -o ./copilot/new_monthly_payment.yaml
```

Copilot: Let's start creating a configuration for your WKO agent.
To begin, could you please describe the functionality you envision for your agent? What task do you want it to perform?
You: Use the interest_rate_lookup tool to find the interest rate, then use the monthly_payment_calc tool to calculate the monthly payment

Copilot: Based on your description, I recommend the following tools for your agent:
1. interest_rate_lookup
2. monthly_payment_calc

These tools are directly mentioned in your description and seem essential for your agent's task.

There are 3 more tools available in the full list. Would you like to see the full list?

Please select the tools you want your agent to use by typing their numbers or names. If you want to use all the recommended tools, you can type 'all'. If you don't want to use any tools, you can type 'none'.

You: 1

Activate Windows
Go to Settings to activate Windows.

1:1 LF UTF-8 4 spaces Python 3.12 (wko-sdk) FRA 1:38 PM 8/21/2024

4. Type 1 and 2 (A).

WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

```
ADK >orchestrate copilot prompt-tune -o ./copilot/new_monthly_payment.yaml
```

Copilot: Let's start creating a configuration for your WKO agent.
To begin, could you please describe the functionality you envision for your agent? What task do you want it to perform?
You: Use the interest_rate_lookup tool to find the interest rate, then use the monthly_payment_calc tool to calculate the monthly payment

Copilot: Based on your description, I recommend the following tools for your agent:
1. interest_rate_lookup
2. monthly_payment_calc

These tools are directly mentioned in your description and seem essential for your agent's task.

There are 3 more tools available in the full list. Would you like to see the full list?

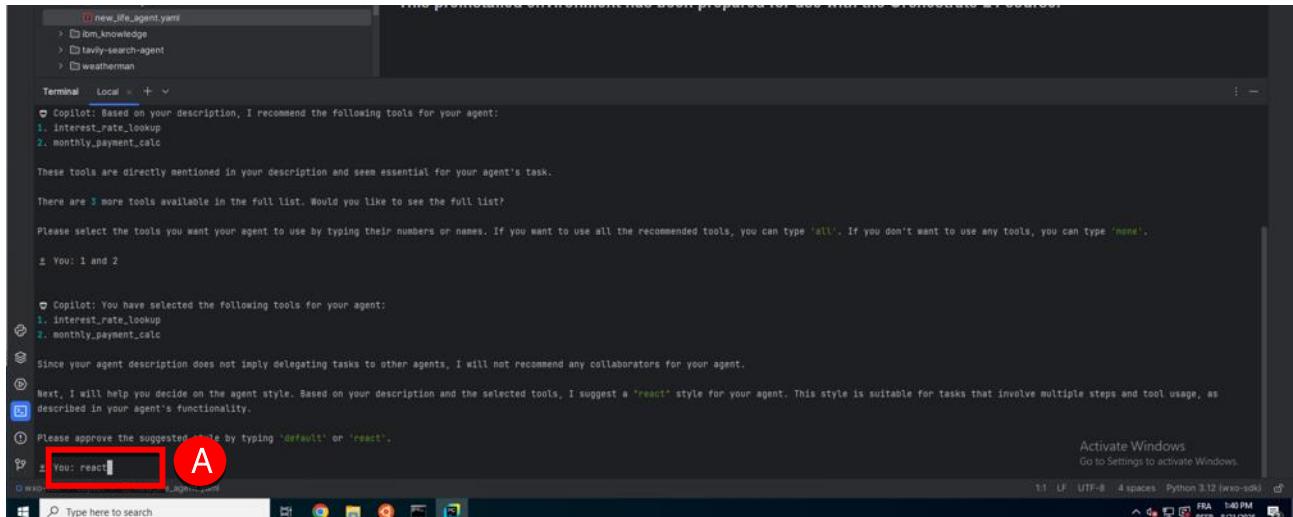
Please select the tools you want your agent to use by typing their numbers or names. If you want to use all the recommended tools, you can type 'all'. If you don't want to use any tools, you can type 'none'.

You: 1 and 2

Activate Windows
Go to Settings to activate Windows.

1:1 LF UTF-8 4 spaces Python 3.12 (wko-sdk) FRA 1:38 PM 8/21/2024

5. Next, copilot suggest the agent type. Type `react` to approve copilot's suggestion (A).



```

new_life_agent.yaml
> tom_knowledge
> tavity-search-agent
> weatherman

Terminal Local + v

Copilot: Based on your description, I recommend the following tools for your agent:
1. interest_rate_lookup
2. monthly_payment_calc

These tools are directly mentioned in your description and seem essential for your agent's task.

There are 3 more tools available in the full list. Would you like to see the full list?

Please select the tools you want your agent to use by typing their numbers or names. If you want to use all the recommended tools, you can type 'all'. If you don't want to use any tools, you can type 'none'.

± You: 1 and 2

Copilot: You have selected the following tools for your agent:
1. interest_rate_lookup
2. monthly_payment_calc

Since your agent description does not imply delegating tasks to other agents, I will not recommend any collaborators for your agent.

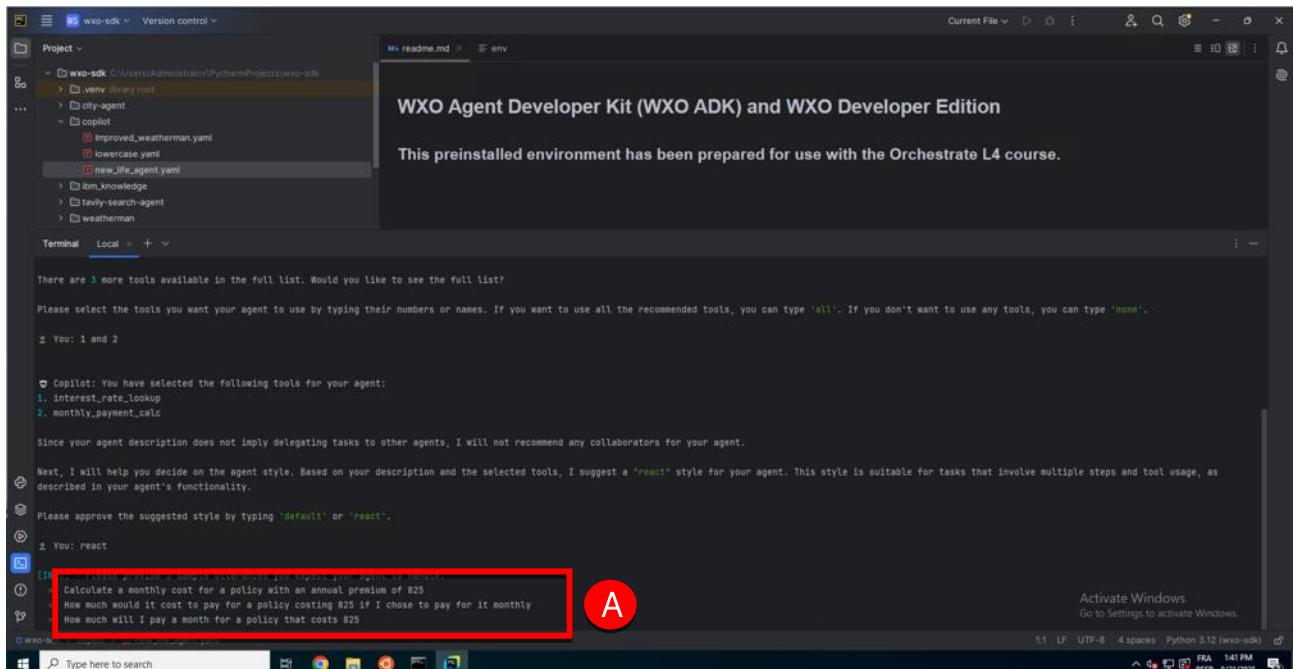
Next, I will help you decide on the agent style. Based on your description and the selected tools, I suggest a "react" style for your agent. This style is suitable for tasks that involve multiple steps and tool usage, as described in your agent's functionality.

Please approve the suggested style by typing 'default' or 'react'.
± You: react

```

6. Next copilot asks for the 3 invocation sentences. You can enter the 3 following sentences on-by-one (A).

Calculate a monthly cost for a policy with an annual premium of 825
 How much would it cost to pay for a policy costing 825 if I chose to pay for it monthly
 How much will I pay a month for a policy that costs 825



```

Project: wxo-sdk Version control: Current File: README.md env

WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

There are 3 more tools available in the full list. Would you like to see the full list?

Please select the tools you want your agent to use by typing their numbers or names. If you want to use all the recommended tools, you can type 'all'. If you don't want to use any tools, you can type 'none'.

± You: 1 and 2

Copilot: You have selected the following tools for your agent:
1. interest_rate_lookup
2. monthly_payment_calc

Since your agent description does not imply delegating tasks to other agents, I will not recommend any collaborators for your agent.

Next, I will help you decide on the agent style. Based on your description and the selected tools, I suggest a "react" style for your agent. This style is suitable for tasks that involve multiple steps and tool usage, as described in your agent's functionality.

Please approve the suggested style by typing 'default' or 'react'.
± You: react

Calculate a monthly cost for a policy with an annual premium of 825
How much would it cost to pay for a policy costing 825 if I chose to pay for it monthly
How much will I pay a month for a policy that costs 825

```

Refine the prompt

Here again the refine sequence is provided as an example. You will follow your own conversation with copilot trying to stick to the recommendation made in this example.

1. The first refinement question from copilot is about the format. We have replied copilot that ‘A simple statement will work’ (A).

The screenshot shows a PyCharm interface with a terminal window open. The terminal content is as follows:

```
1. Calculate a monthly cost for a policy with an annual premium of $25
2. How much would it cost to pay for a policy costing $25 if I chose to pay for it monthly
3. How much will I pay a month for a policy that costs $25

Copilot: Based on the task description, it seems like we need to use the interest_rate_lookup tool to find the interest rate and then use the monthly_payment_calc tool to calculate the monthly payment.

Here's an initial prompt:

**Role**
You are an agent that calculates monthly costs for policies with given annual premiums.

**Tool Usage Guidelines**
1. Before you call a tool, make sure you have all the required tool parameters.
2. Do NOT assume any tool parameters
3. Do NOT pass parameters to the tool that are not explicitly defined
4. Do NOT call the same tool multiple times with the same parameters

**How To Use Tools**
- When the user wants to calculate the monthly cost for a policy, first call the interest_rate_lookup tool to retrieve the interest rate using the annual premium.
- Then, use the monthly_payment_calc tool to calculate the monthly payment using the annual premium and the retrieved interest rate.

Q: How would you like to format the output of the monthly_payment calculation? Would you like it to be a simple statement, a table, or something else?
A: You: A simple statement will work
```

A red box highlights the response "You: A simple statement will work" with a label "A" in a circle. The terminal window has a dark background with light-colored text.

2. We are now good. Let’s notify copilot we are satisfied (A) to stop the refinement.

The screenshot shows a PyCharm interface with a terminal window open. The terminal content is as follows:

```
4. Do NOT call the same tool multiple times with the same parameters

**How To Use Tools**
- When the user provides an annual premium, call the interest_rate_lookup tool to find the interest rate.
- Next, use the monthly_payment_calc tool to calculate the monthly payment, passing the annual premium and the interest rate found in the previous step.
- Format the output as a simple statement, e.g., *The monthly cost for the policy is $X*.

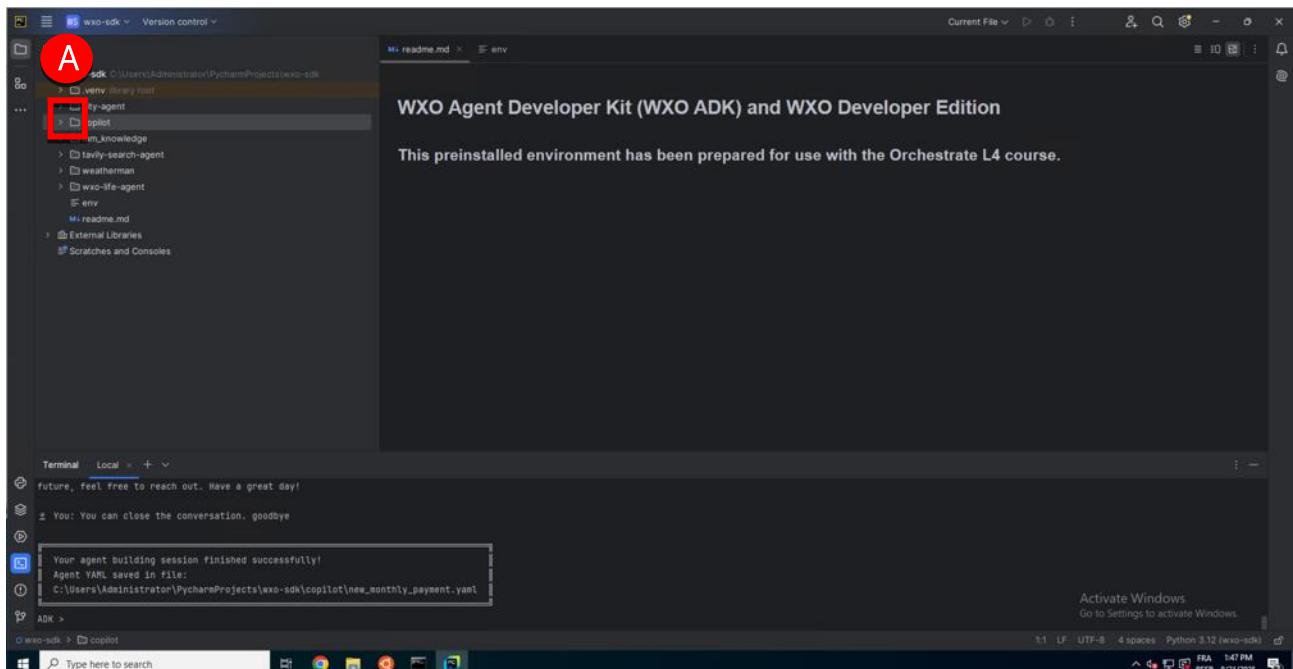
Q: Are you interested in continuing to refine the prompt, or is this sufficient for your needs?
A: You: Stop
```

A red box highlights the response "You: Stop" with a label "A" in a circle. The terminal window has a dark background with light-colored text.

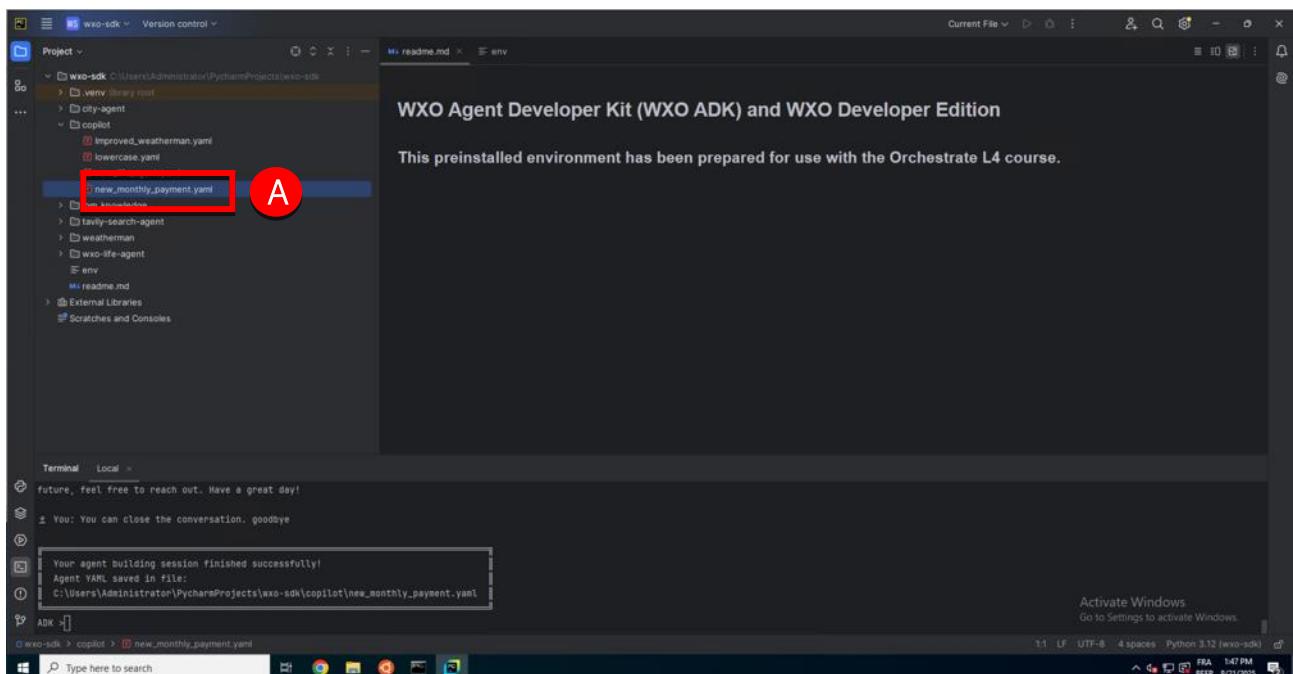
Note: Copilot will generate the agent file only after closing the conversation properly.

Review the agent

1. Expand the copilot folder (A).



2. Double-click on new_monthly_payment.yaml (A) to open the file.



3. Review the generated agent description.

- The name copilot gave to your agent (A)
- The agent description generated by copilot (B).
- The agent style ‘react’ as discussed with copilot (C).
- The instructions generated from you examples and refinement process (D).
- The use of the 2 tools tool as specified to copilot (E).

The screenshot shows a PyCharm interface with a project named 'wxo-sdk'. The 'new_monthly_payment.yaml' file is open in the editor. The code defines a native agent named 'interest_rate_monthly_payment' with a 'react' style. It includes a description, context access, variables, and a specified LLM. The 'collaborators' section lists 'interest_rate_lookup' and 'monthly_payment_calc'. The terminal below shows a successful build message.

```

spec_version: v1
type: native
name: interest_rate_monthly_payment
description: Use the interest_rate_lookup tool to find the interest rate, then use the monthly_payment_calc tool to calculate the monthly payment
context_access_enabled: true
context_variables: []
llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct
style: react
instructions: |
  You are an agent that calculates monthly costs for policies with given annual premiums.

  **Tool Usage Guidelines**
  1. Before you call a tool, make sure you have all the required tool parameters.
  2. Do NOT assume any tool parameters
  3. Do NOT pass parameters to the tool that are not explicitly defined
  4. Do NOT call the same tool multiple times with the same parameters

  **How To Use Tools**
  - When the user wants to calculate the monthly cost for a policy, first call the interest_rate_lookup tool to retrieve the interest rate using the annual premium. Then, use the monthly_payment_calc tool to calculate the monthly payment using the annual premium and the retrieved interest rate.

  collaborators: []
  - interest_rate_lookup
  - monthly_payment_calc

```

Your agent building session finished successfully!
Agent YAML saved in file:
C:\Users\Administrator\PycharmProjects\wxo-sdk\copilot\new_monthly_payment.yaml

Part 3: Create a managing agent with collaborators

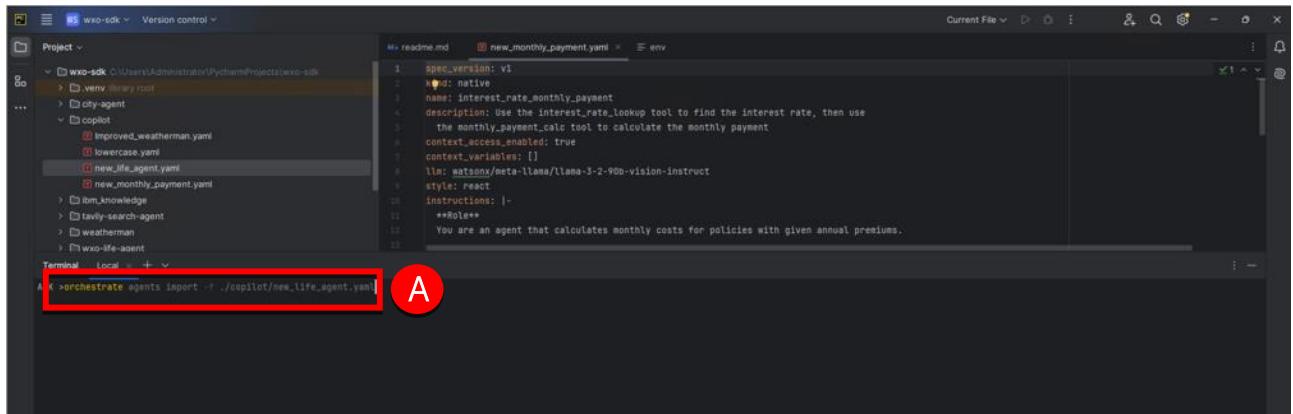
In this last part, you will use copilot to create a managing agent able to work with the different collaborators you have previously created.

Before starting copilot, you must import the two new agents (life_agent and monthly_payment) into the environment so that copilot can analyze them when defining the manager agent.

Import the new agents.

- Type the following command to import the new_life_agent into your environment (A).

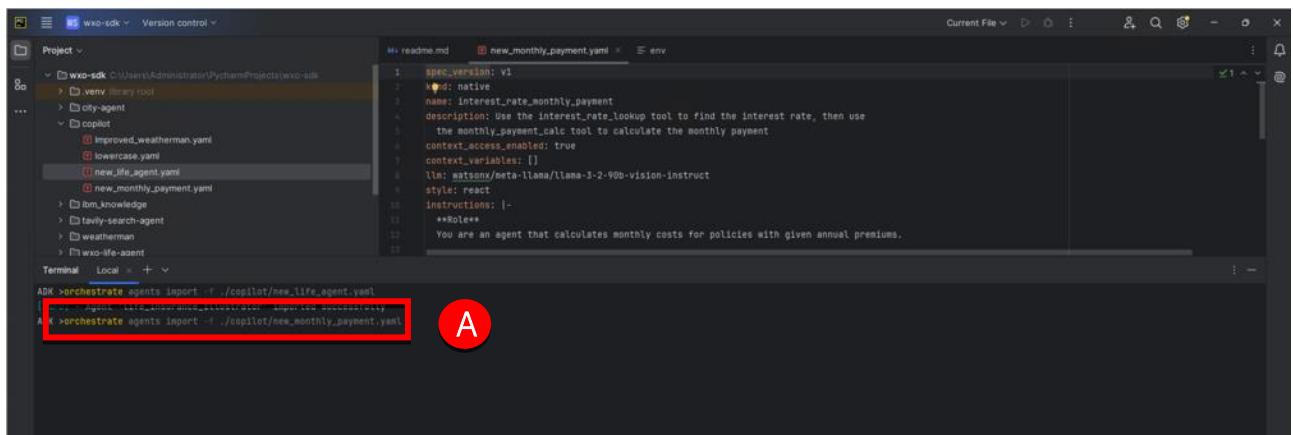
```
orchestrate agents import -f ./copilot/new_life_agent.yaml
```



A screenshot of the PyCharm IDE interface. The left sidebar shows a project structure with a file named 'new_monthly_payment.yaml' selected. The main editor window displays the YAML content for this file. Below the editor is a terminal window. In the terminal window, the command `orchestrate agents import -f ./copilot/new_life_agent.yaml` is typed and highlighted with a red box. A red circle with the letter 'A' is placed over the terminal window, indicating the step to follow.

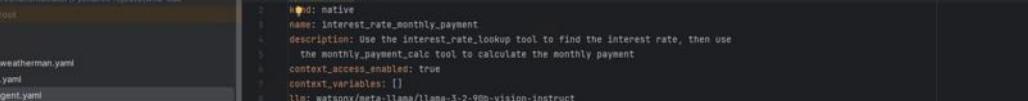
- Type the following command to import the new_monthly_payment agent into your environment (A).

```
orchestrate agents import -f ./copilot/new_monthly_payment.yaml
```



A screenshot of the PyCharm IDE interface, similar to the previous one. The left sidebar shows the same project structure. The terminal window at the bottom has two commands typed: `ADK >orchestrate agents import -f ./copilot/new_life_agent.yaml` and `A < >orchestrate agents import -f ./copilot/new_monthly_payment.yaml`. The second command is highlighted with a red box and has a red circle with the letter 'A' placed over it.

3. Type 'orchestrate agents list' to review your available agents in your environment (A).



The screenshot shows a terminal window with the following command history:

```
ADM >orchestrate agents import -f ./copilot/new_life_agent.yaml
[INFO] - Agent 'life_insurance_illustrator' imported successfully
ADM >orchestrate agents import -f ./copilot/new_monthly_payment.yaml
[INFO] - Agent 'new_monthly_payment' imported successfully
ADM >orchestrate agents list
```

4. Make sure the two new agents are in the list (A).

Project

- wxo-sdk
- venv
- city-agent
- copilot
 - Improved_weatherman.yaml
 - lowercase.yaml
 - new_life_agent.yaml
 - new_monthly_payment.yaml
- tim_knowledge
- tivity-search-agent
- weatherman
- wxo-life-agent

Terminal Local × +

```
ADK >orchestrate agents import -f ./copilot/new_monthly_payment.yaml
[INFO] - Agent 'interest_rate_monthly_payment' imported successfully
ADK >orchestrate agents list
```

Name	Description	LLM	Style	Collaborators	Tools	Knowledge Base	ID
interest_rate_monthly_payment	use the interest_rate_lookup tool to find the interest rate, then use the monthly_payment_calc tool to calculate the monthly payment	watsonx/meta-llama/llama-3-2-90B-v1	react	sion-instruct	interest_rate_lookup, monthly_payment_calc		0de8e95c-01e2-4bc2-bf46-f87528c150e6
life_insurance_illustrator	Provide a life insurance illustration that includes the annual premium of the policy.	watsonx/meta-llama/llama-3-2-90B-v1	default	sion-instruct	life_quote		4b9ad0de-3bb1-4000-8bbb-f9a1f77ad8ef
weatherman	You provide the current temperature for well known cities.	watsonx/meta-llama/llama-3-2-90B-v1	react	sion-instruct	forecast		ddab6b75-4f53-4b4c-bbbb-b1978029fd7d
text.lowercase	An agent that takes a word or sentence as input and returns it in all lowercase letters.	watsonx/meta-llama/llama-3-2-90B-v1	default	sion-instruct			d51b9459-30f0-4f46-9b3f-550c033e3e7e

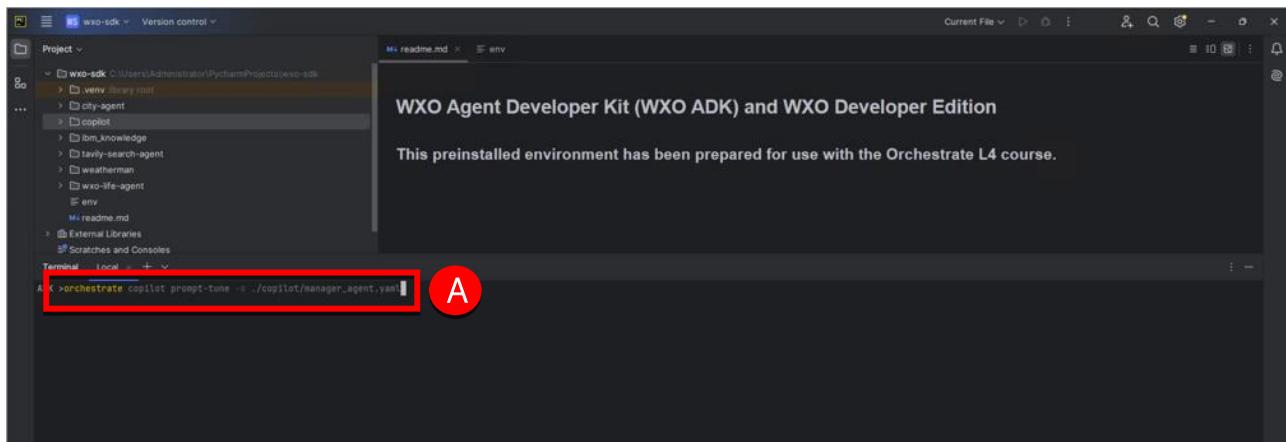
A

You are now ready to start building the manager agent.

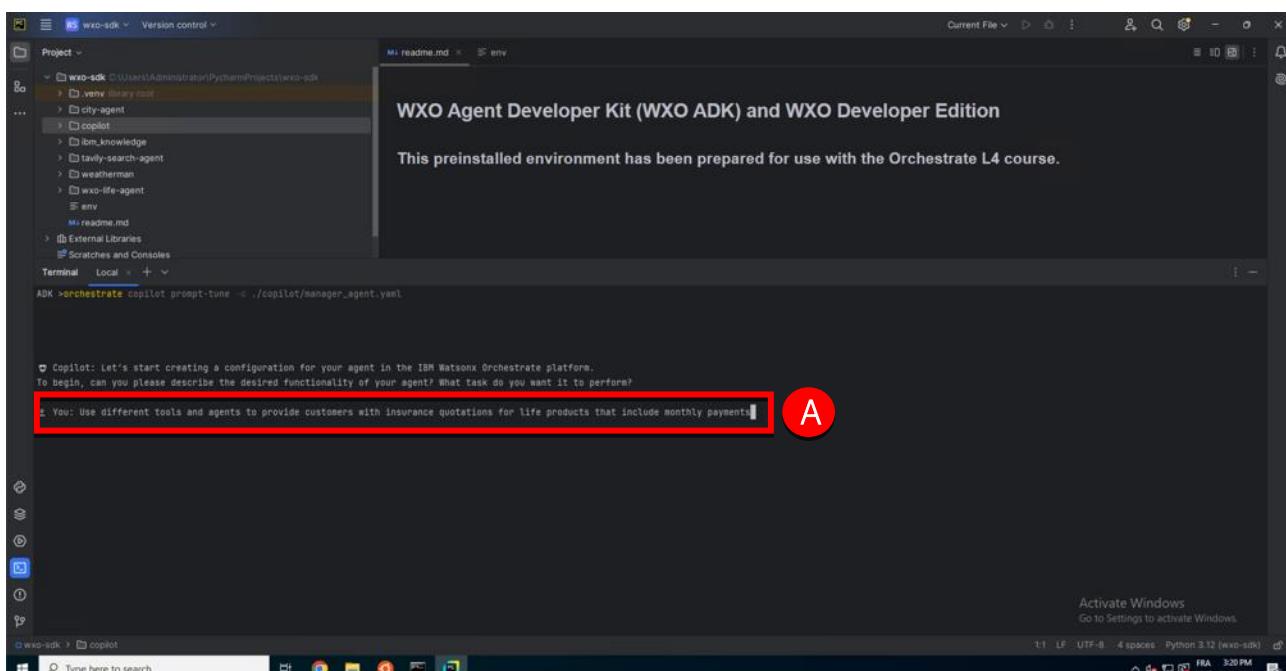
Start copilot to create the agent

- Type the following command to start the manager agent creation with copilot (A).

```
orchestrate copilot prompt-tune -o ./copilot/manager_agent.yaml
```



- Type 'Use different tools and agents to provide customers with insurance quotations for life products that include monthly payments' to start defining the agent definition' (A).



Copilot has identified that you are working on a manager agent. It directly suggests the use of collaborators that can help to get the work done. Copilot just asks you for the final decision.

3. Type 1 and 2 (A) to select the 2 agents.

The screenshot shows the PyCharm IDE interface with the following details:

- Project Tree:** Shows the `wko-sdk` project structure under `C:\Users\Administrator\PycharmProjects\wko-sdk`. It includes subfolders like `venv`, `library`, `agent`, `copilot`, `ibm_knowledge`, `tavly-search-agent`, `weatherman`, `wko-life-agent`, and `env`. There are also files `README.md` and `External Libraries`.
- Terminal:** The terminal tab is active, showing the command `ADK >orchestrate copilot prompt-tune -c ./copilot/manager_agent.yaml`.
- Output:** The terminal output shows a conversation with a copilot:
 - Copilot asks for a configuration description.
 - The user responds with a desire to provide insurance quotations for life products.
 - Copilot suggests selecting collaborators first.
 - The user lists recommended collaborators: `interest_rate_monthly_payment` and `life_insurance_illustrator`.
 - The user is prompted to select collaborators by number or name.A red circle labeled "A" highlights the input field where the user has typed `1 You: I and 2`.

Note: If copilot suggests other choice, just let the tool know the agents you need for this work. If no collaborators are suggested, exit and restart, and mention the collaboration agents you wish to use in your initial description.

- When copilot comes to the tool selection, you can say ‘none’ (A) of the tools since the agents are already designed to use them.

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** wko-sdk
- File:** Mu README.md
- Code Editor:** Content of README.md:

```
WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

There are 3 more agents in the full list. Would you like to see it?

Please select the collaborators you want your agent to use by indicating their numbers or names.

1 You: 1 and 2

Copilot: You have selected the following collaborators:
1. interest_rate_monthly_payment
2. life_insurance_illustrator

Now, let's move on to selecting tools.

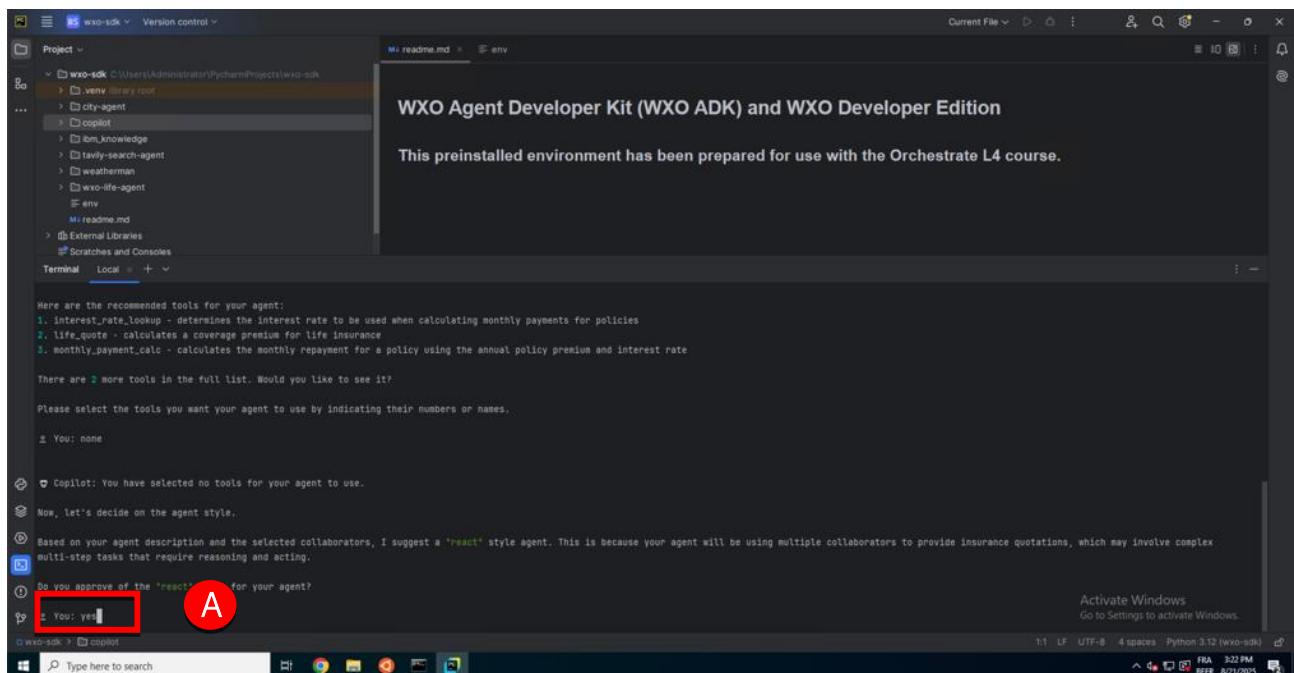
Here are the recommended tools for your agent:
1. interest_rate_lookup - determines the interest rate to be used when calculating monthly payments for policies
2. life_quote - calculates a coverage premium for life insurance
3. monthly_payment_calc - calculates the monthly repayment for a policy using the annual policy premium and interest rate

There are 3 more tools in the full list. Would you like to see it?

Please select the tools you want your agent to use by indicating their numbers or names.

1 You: none
```
- Bottom Status Bar:** Activate Windows, Go to Settings to activate Windows.
- Bottom Icons:** PyCharm icons for file operations like Open, Save, Find, and Run.

5. Confirm the choice of react for the agent to copilot (A).



```

WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

Here are the recommended tools for your agent:
1. interest_rate_lookup - determines the interest rate to be used when calculating monthly payments for policies
2. life_quote - calculates a coverage premium for life insurance
3. monthly_payment_calc - calculates the monthly repayment for a policy using the annual policy premium and interest rate

There are 2 more tools in the full list. Would you like to see it?

Please select the tools you want your agent to use by indicating their numbers or names.

You: none

Copilot: You have selected no tools for your agent to use.

Now, let's decide on the agent style.

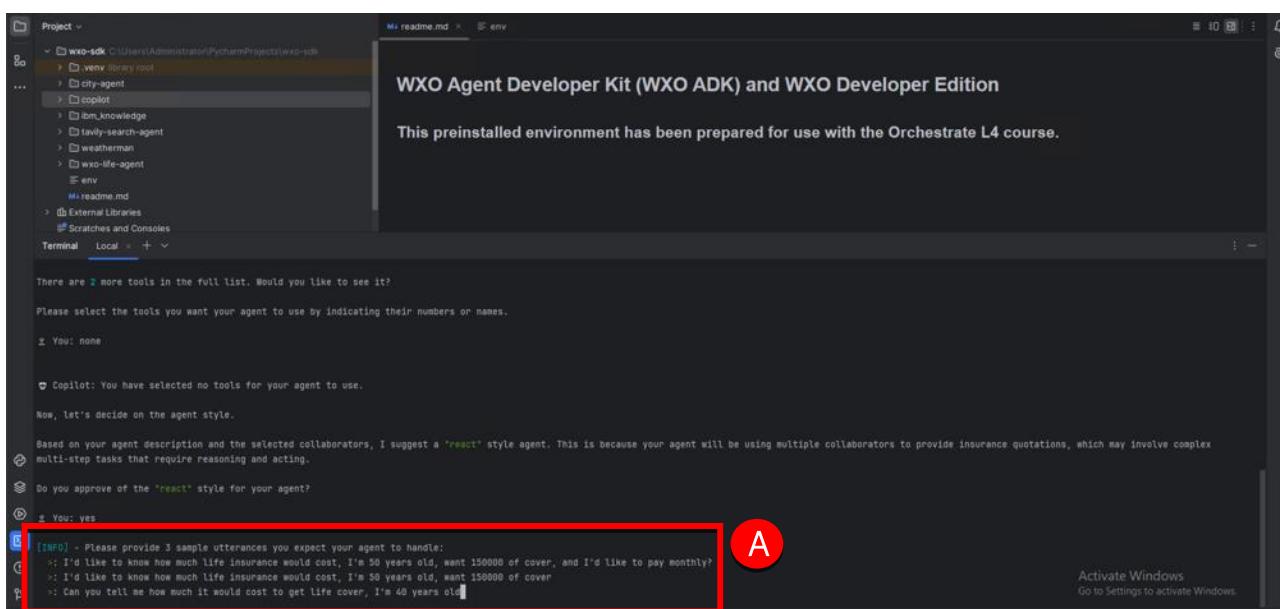
Based on your agent description and the selected collaborators, I suggest a 'react' style agent. This is because your agent will be using multiple collaborators to provide insurance quotations, which may involve complex multi-step tasks that require reasoning and acting.

Do you approve of the 'react' style for your agent?

You: yes
  
```

6. Enter the 3 invocation sentences one-by-one when prompted (A)

I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, and I'd like to pay monthly?
I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover
Can you tell me how much it would cost to get life cover, I'm 40 years old



```

WKO Agent Developer Kit (WKO ADK) and WKO Developer Edition

This preinstalled environment has been prepared for use with the Orchestrate L4 course.

There are 2 more tools in the full list. Would you like to see it?

Please select the tools you want your agent to use by indicating their numbers or names.

You: none

Copilot: You have selected no tools for your agent to use.

Now, let's decide on the agent style.

Based on your agent description and the selected collaborators, I suggest a 'react' style agent. This is because your agent will be using multiple collaborators to provide insurance quotations, which may involve complex multi-step tasks that require reasoning and acting.

Do you approve of the 'react' style for your agent?

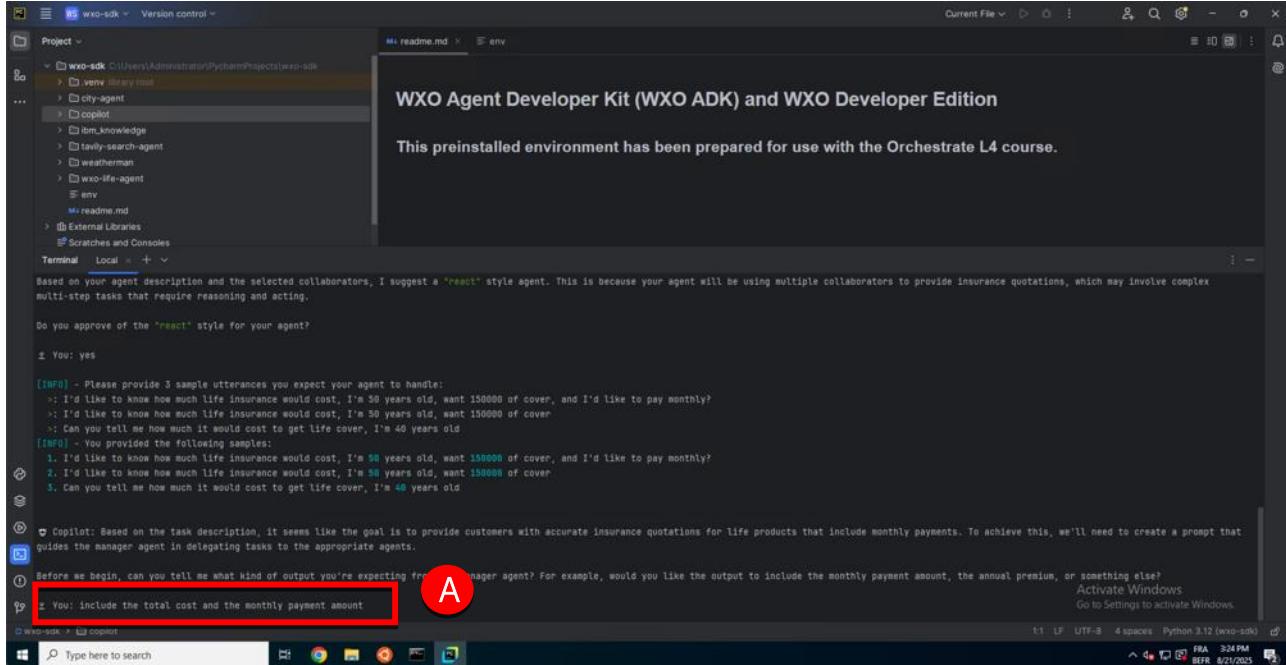
You: yes
  
```

[INFO] - Please provide 3 sample utterances you expect your agent to handle:
You: I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, and I'd like to pay monthly?
You: I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover
You: Can you tell me how much it would cost to get life cover, I'm 40 years old?

Refine the prompt

Here again the refine sequence is provided as an example. You will follow your own conversation with copilot trying to stick to the recommendation made in this example.

1. Instruct copilot that you want to ‘include the total cost and the monthly payment amount’ in the response (A).



The screenshot shows a PyCharm interface with a terminal window open. The terminal content is as follows:

```
Based on your agent description and the selected collaborators, I suggest a "react" style agent. This is because your agent will be using multiple collaborators to provide insurance quotations, which may involve complex multi-step tasks that require reasoning and acting.

Do you approve of the "react" style for your agent?

# You: yes

[INFO] - Please provide 3 sample utterances you expect your agent to handle:
> I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, and I'd like to pay monthly?
> I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover
> Can you tell me how much it would cost to get life cover, I'm 40 years old

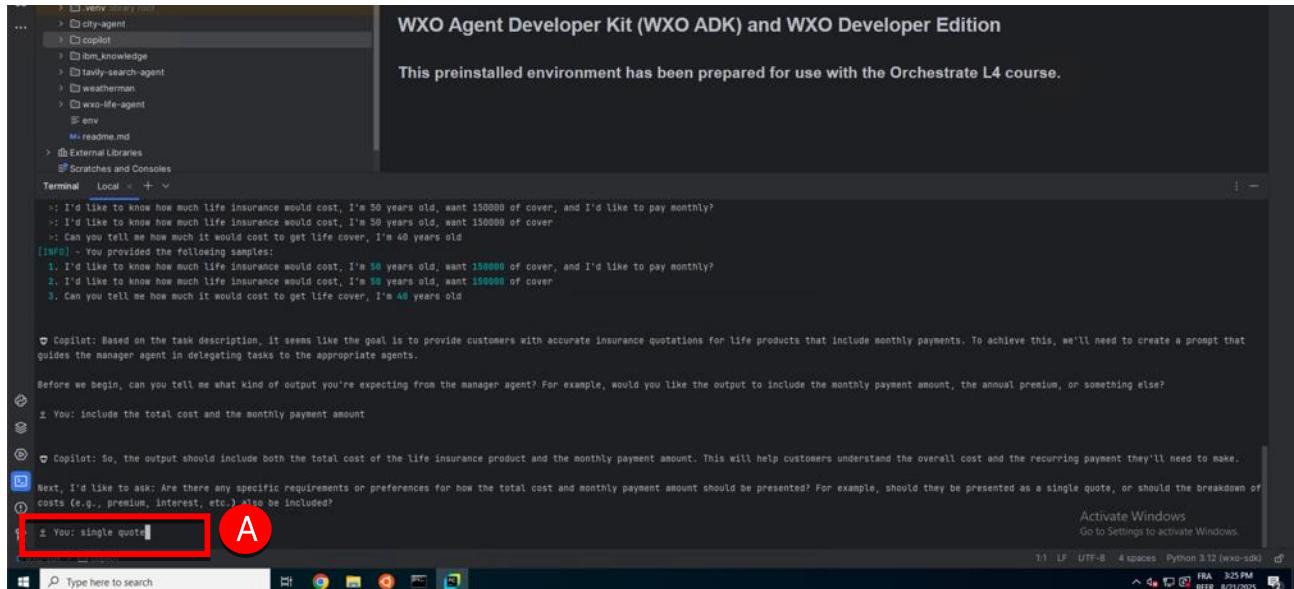
[INFO] - You provided the following samples:
1. I'd like to know how much life insurance would cost, I'm 50 years old, want $150000 of cover, and I'd like to pay monthly?
2. I'd like to know how much life insurance would cost, I'm 50 years old, want $150000 of cover
3. Can you tell me how much it would cost to get life cover, I'm 40 years old

④ Copilot: Based on the task description, it seems like the goal is to provide customers with accurate insurance quotations for life products that include monthly payments. To achieve this, we'll need to create a prompt that guides the manager agent in delegating tasks to the appropriate agents.

① Before we begin, can you tell me what kind of output you're expecting from the manager agent? For example, would you like the output to include the monthly payment amount, the annual premium, or something else?
② You: include the total cost and the monthly payment amount
```

A red box highlights the user input "x You: include the total cost and the monthly payment amount". A red circle with the letter "A" is positioned over the terminal window, indicating the point of interest.

2. We were asked about the output type, your response may be different. We have selected ‘single quote’ (A) for the output.



```

.../wxo-sdk> venv library-test
> city-agent
> copilot
> ltm_knowledge
> tadv-search-agent
> weatherman
> wxo-life-agent
> env
> README.md
> External Libraries
> Scratches and Consoles
Terminal Local + 
> I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, and I'd like to pay monthly?
> I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover
> Can you tell me how much it would cost to get life cover, I'm 40 years old
[INFO] - You provided the following samples:
1. I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, and I'd like to pay monthly?
2. I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover
3. Can you tell me how much it would cost to get life cover, I'm 40 years old

Copilot: Based on the task description, it seems like the goal is to provide customers with accurate insurance quotations for life products that include monthly payments. To achieve this, we'll need to create a prompt that guides the manager agent in delegating tasks to the appropriate agents.

Before we begin, can you tell me what kind of output you're expecting from the manager agent? For example, would you like the output to include the monthly payment amount, the annual premium, or something else?
You: include the total cost and the monthly payment amount

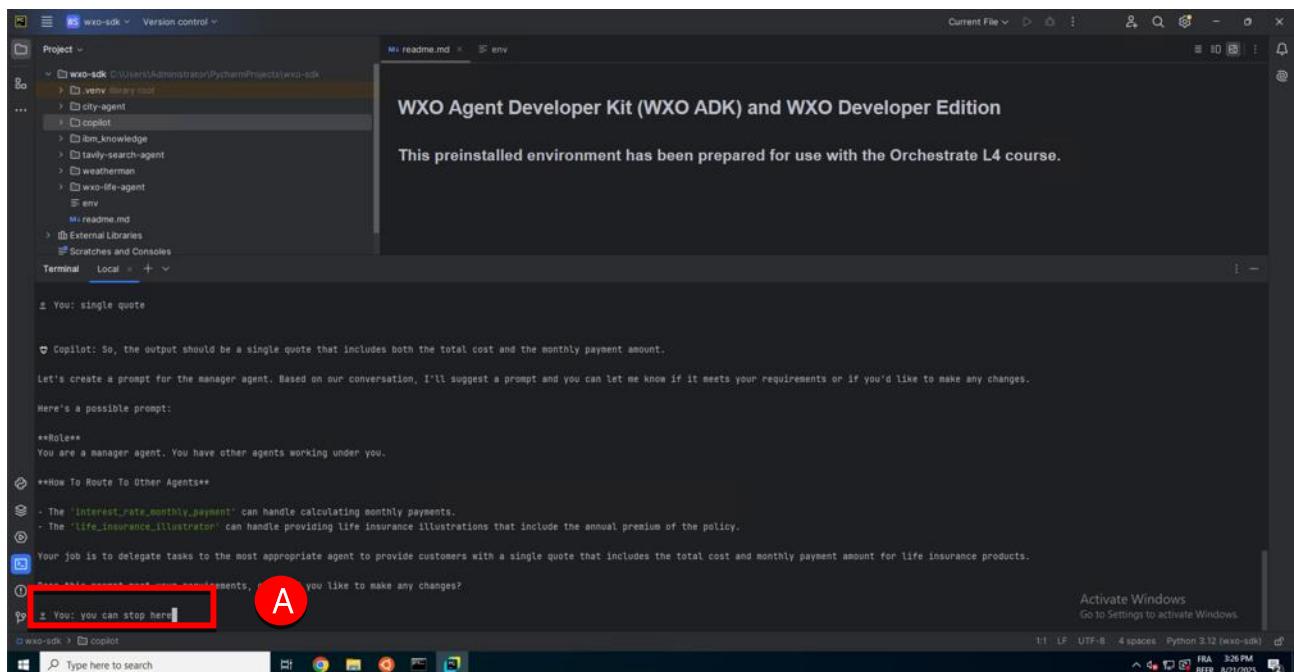
Copilot: So, the output should include both the total cost of the life insurance product and the monthly payment amount. This will help customers understand the overall cost and the recurring payment they'll need to make.

Next, I'd like to ask: Are there any specific requirements or preferences for how the total cost and monthly payment amount should be presented? For example, should they be presented as a single quote, or should the breakdown of costs (e.g., premium, interest, etc.) be included?

1 You: single quote

```

3. Let copilot know when you are done with the refinement (A).



```

.../wxo-sdk> venv library-test
> city-agent
> copilot
> ltm_knowledge
> tadv-search-agent
> weatherman
> wxo-life-agent
> env
> README.md
> External Libraries
> Scratches and Consoles
Terminal Local + 
> You: single quote

Copilot: So, the output should be a single quote that includes both the total cost and the monthly payment amount.

Let's create a prompt for the manager agent. Based on our conversation, I'll suggest a prompt and you can let me know if it meets your requirements or if you'd like to make any changes.

Here's a possible prompt:

**Role**
You are a manager agent. You have other agents working under you.

**How To Route To Other Agents**
- The 'interest_rate_monthly_payment' can handle calculating monthly payments.
- The 'life_insurance_illustrator' can handle providing life insurance illustrations that include the annual premium of the policy.

Your job is to delegate tasks to the most appropriate agent to provide customers with a single quote that includes the total cost and monthly payment amount for life insurance products.

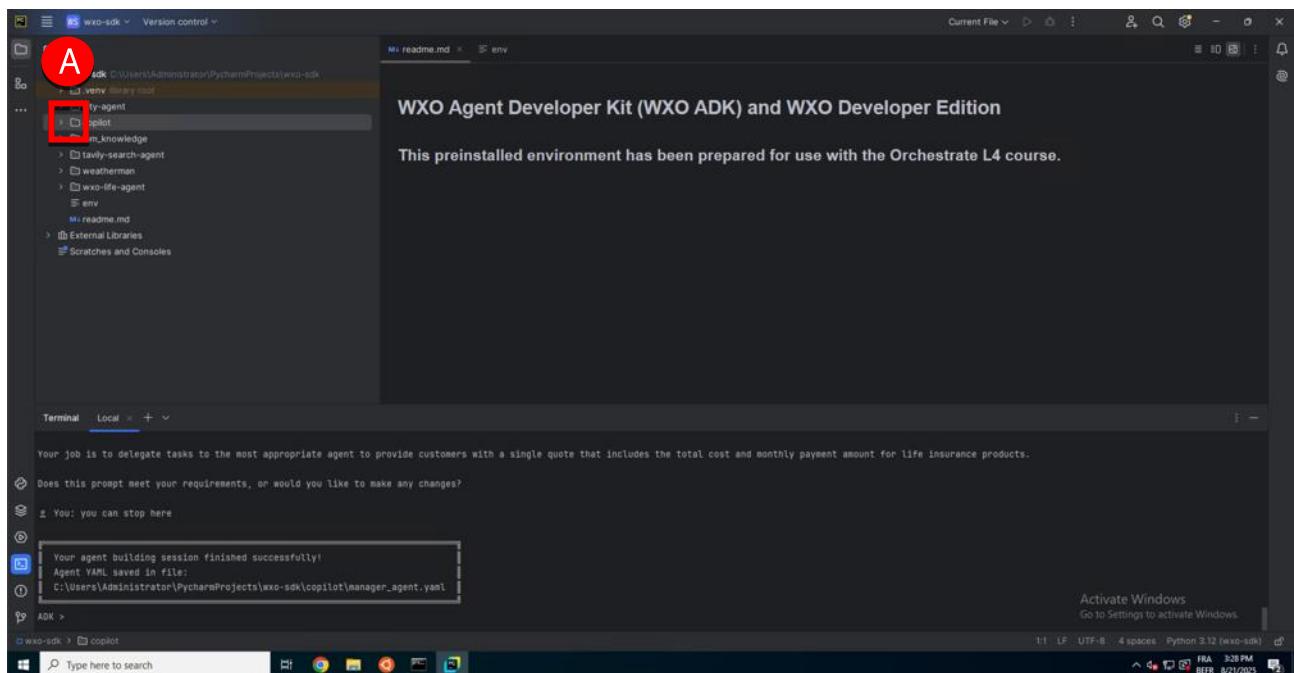
2 You: you can stop here

```

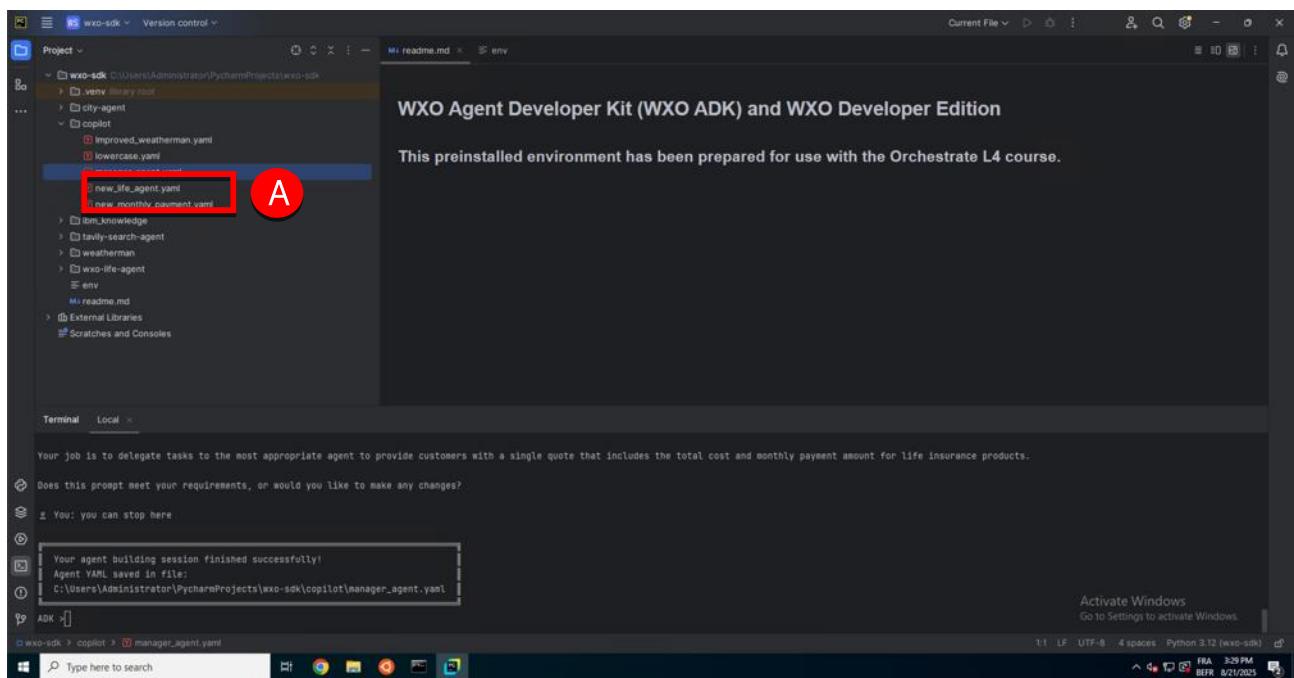
Review the agent

You can now review the generated file.

1. Expand the copilot folder (A).



2. Double-click on manager_agent.yaml (A) to open the file.



3. Review the generated agent description.

- The name copilot gave to your agent (A)
- The agent description generated by copilot (B).
- The agent style ‘react’ as discussed with copilot (C).
- The instructions generated from you examples and refinement process (D).
- The use of the 2 agent tools as specified to copilot (E).

The screenshot shows a PyCharm IDE with a project named 'wxo-sdk'. The 'Project' tool window on the left shows a directory structure with 'copilot' containing several YAML files: 'Improved_weatherman.yaml', 'lowercase.yaml', 'manager_agent.yaml', 'new_life_agent.yaml', and 'new_monthly_payment.yaml'. The 'manager_agent.yaml' file is open in the main editor, showing its contents. The terminal at the bottom shows the command 'wko build' being run and its successful completion.

```

spec_version: v1
id: native
name: life_insurance_quoter
description: Use different tools and agents to provide customers with insurance quotes
for life products that include monthly payments
context_access_enabled: true
context_variables: []
l1e: watsonx/meta-llama/llama-3-2-90b-vision-instruct
style: react
instructions:
  **Role**
  You are a manager agent. You have other agents working under you.
  **How To Route To Other Agents**
  - The 'interest_rate_monthly_payment' can handle calculating monthly payments.
  - The 'life_insurance_illustrator' can handle providing life insurance illustrations that include the annual premium of the policy.
  Your job is to delegate tasks to the most appropriate agent to provide customers with a single quote that includes the total cost and monthly payment amount.
  - interest_rate_monthly_payment
  - life_insurance_illustrator
knowledge_base: []

Terminal Local < + 
You: You can stop here
Your agent building session finished successfully!
Agent YAML saved in file:
C:\users\Administrator\PyCharmProjects\wxo-sdk\copilot\manager_agent.yaml
OK >

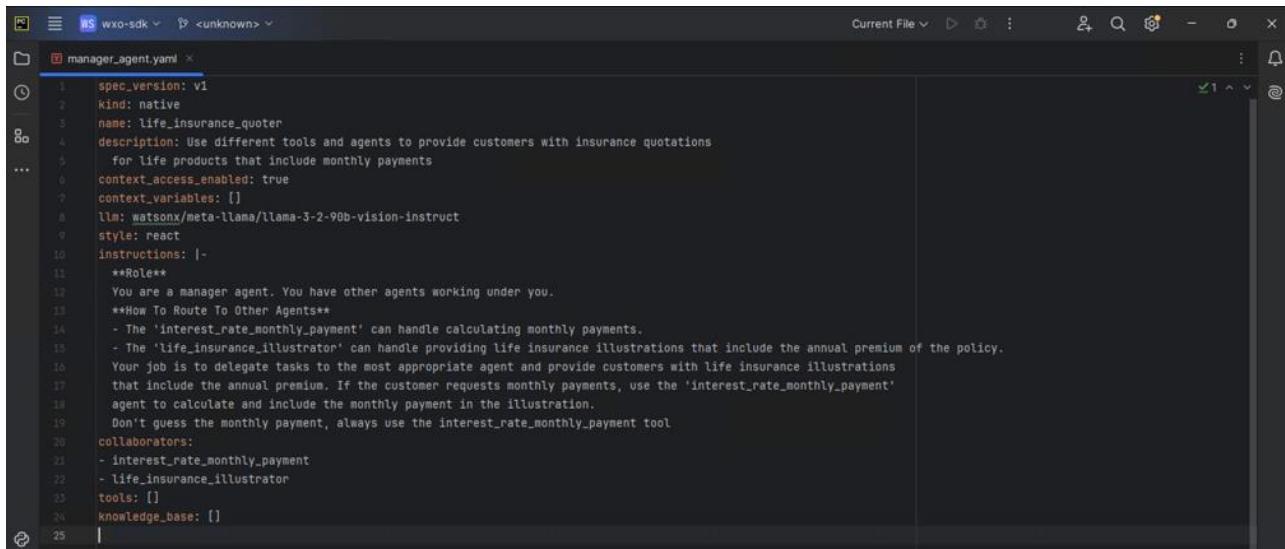
```

4. The agent definition created by copilot has saved us some time, but in its current state it would not produce the results correctly. Further instructions must be provided to ensure the tools are used correctly. Update the instructions to include the following additional tool guidance:

Your job is to delegate tasks to the most appropriate agent and provide customers with life insurance illustrations that include the annual premium. If the customer requests monthly payments, use the 'interest_rate_monthly_payment' agent to calculate and include the monthly payment in the illustration.

Don't guess the monthly payment always use the interest_rate_monthly_payment tool.

Your updated agent should resemble the example below.



The screenshot shows a code editor window with the file 'manager_agent.yaml' open. The file contains YAML configuration for a manager agent. Key sections include 'spec_version: v1', 'kind: native', 'name: life_insurance_quoter', and 'description: Use different tools and agents to provide customers with insurance quotations for life products that include monthly payments'. The 'instructions' section provides a detailed role description for the manager agent, mentioning its responsibility to delegate tasks to other agents and handle monthly payments. It also lists 'collaborators' such as 'interest_rate_monthly_payment' and 'life_insurance_illustrator'.

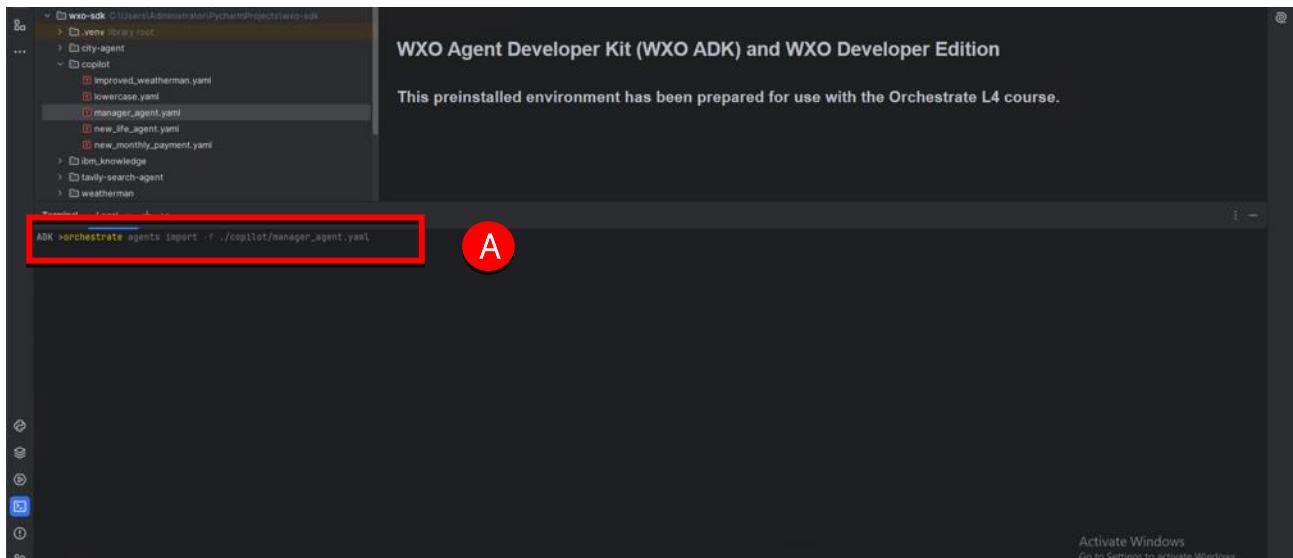
```
spec_version: v1
kind: native
name: life_insurance_quoter
description: Use different tools and agents to provide customers with insurance quotations for life products that include monthly payments
...
context_access_enabled: true
context_variables: []
llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct
style: react
instructions: |-
    **Role**
    You are a manager agent. You have other agents working under you.
    **How To Route To Other Agents**
    - The 'interest_rate_monthly_payment' can handle calculating monthly payments.
    - The 'life_insurance_illustrator' can handle providing life insurance illustrations that include the annual premium of the policy.
    Your job is to delegate tasks to the most appropriate agent and provide customers with life insurance illustrations that include the annual premium. If the customer requests monthly payments, use the 'interest_rate_monthly_payment' agent to calculate and include the monthly payment in the illustration.
    Don't guess the monthly payment, always use the interest_rate_monthly_payment tool
collaborators:
- interest_rate_monthly_payment
- life_insurance_illustrator
tools: []
knowledge_base: []
```

In this example it was necessary to override the LLMs natural tendency to guess a monthly payment just by dividing the annual premium by 12.

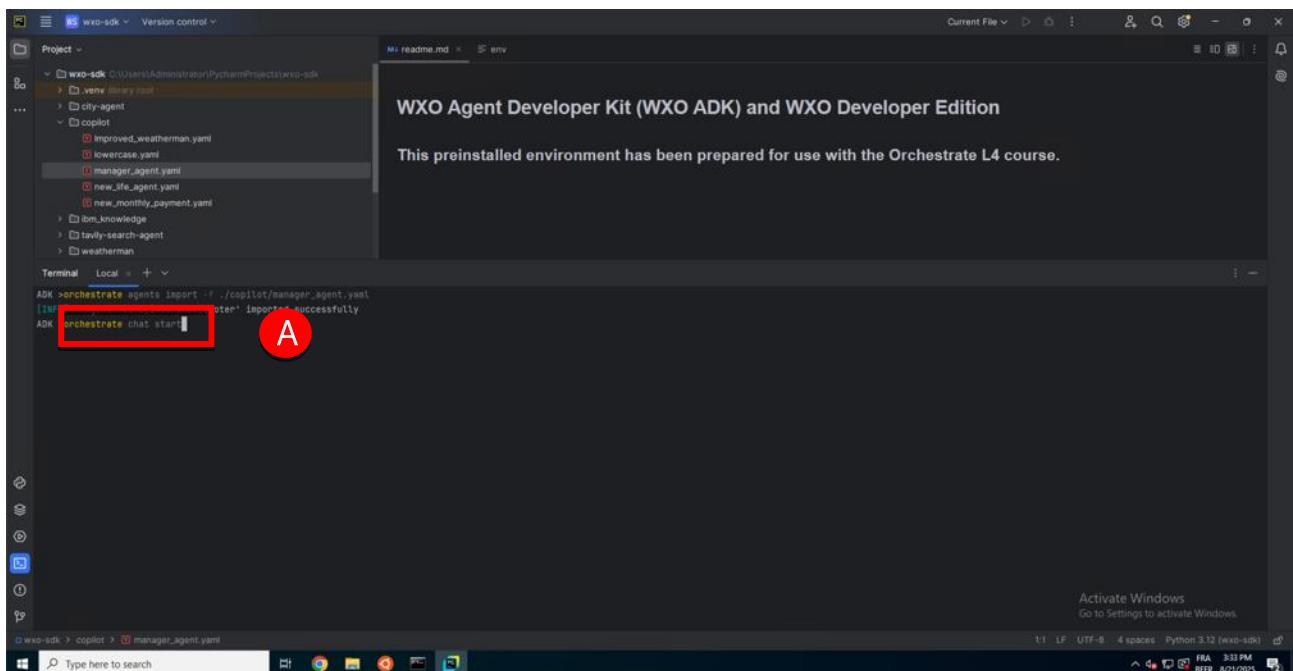
Preview the agent

1. Enter the following command (A) to import your agent:

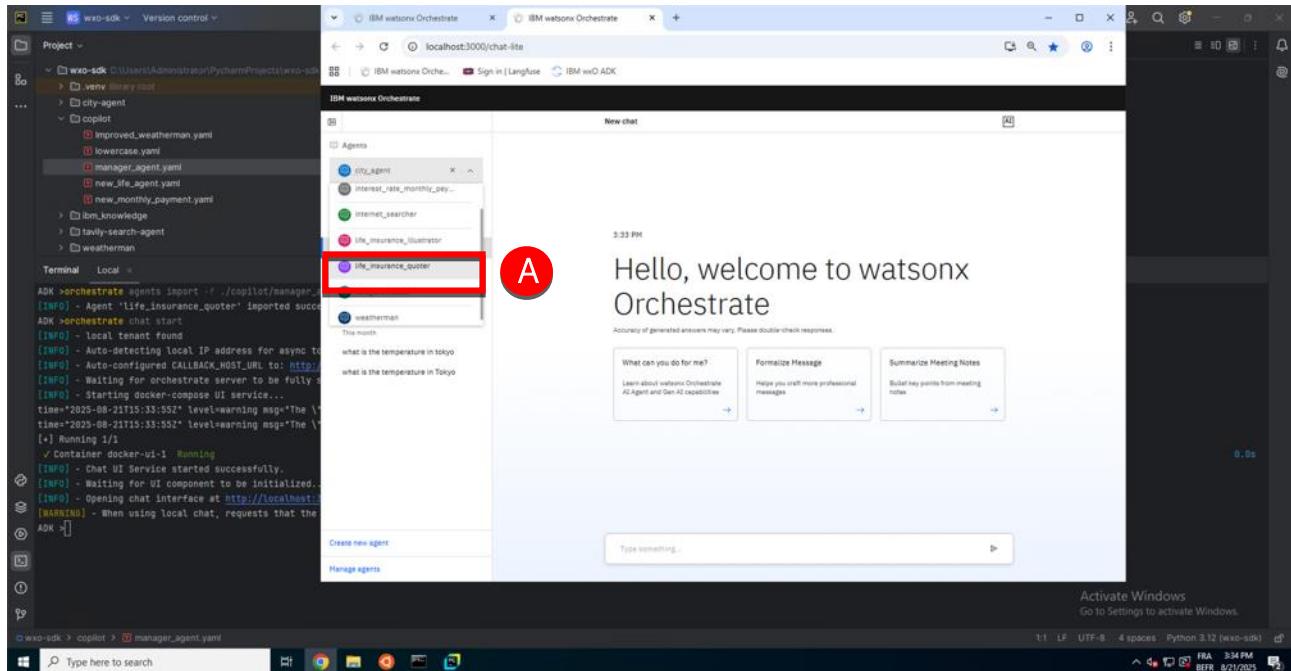
```
orchestrate agents import -f ./copilot/manager_agent.yaml
```



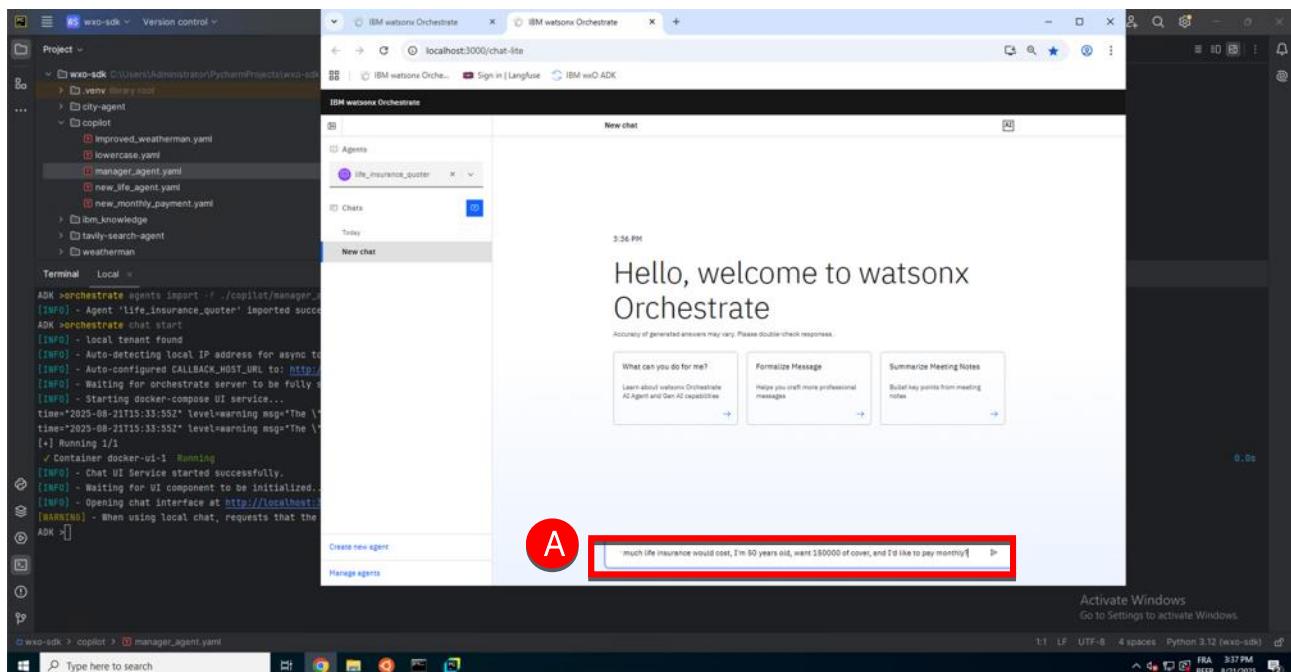
2. Type 'orchestrate chat start' (A) to start the chat interface.



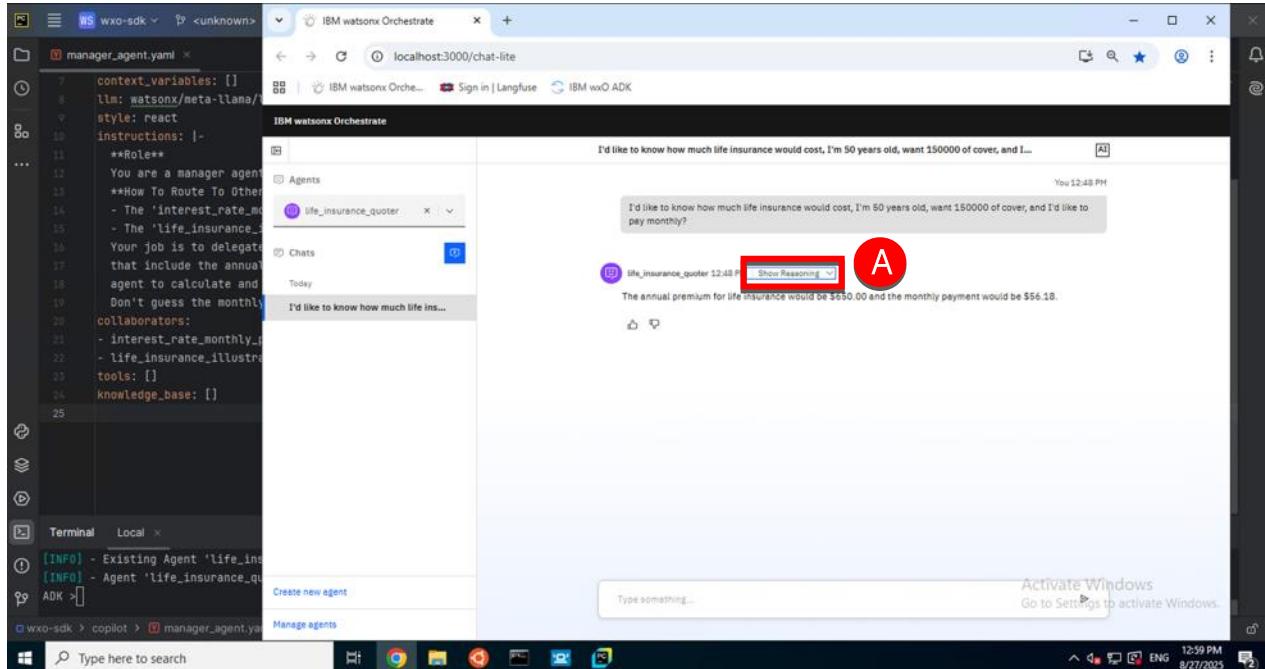
3. Select the `life_insurance_quoter` (use the name that corresponds to the one generated in your yaml file) agent from the agent selector drop-down menu (A).



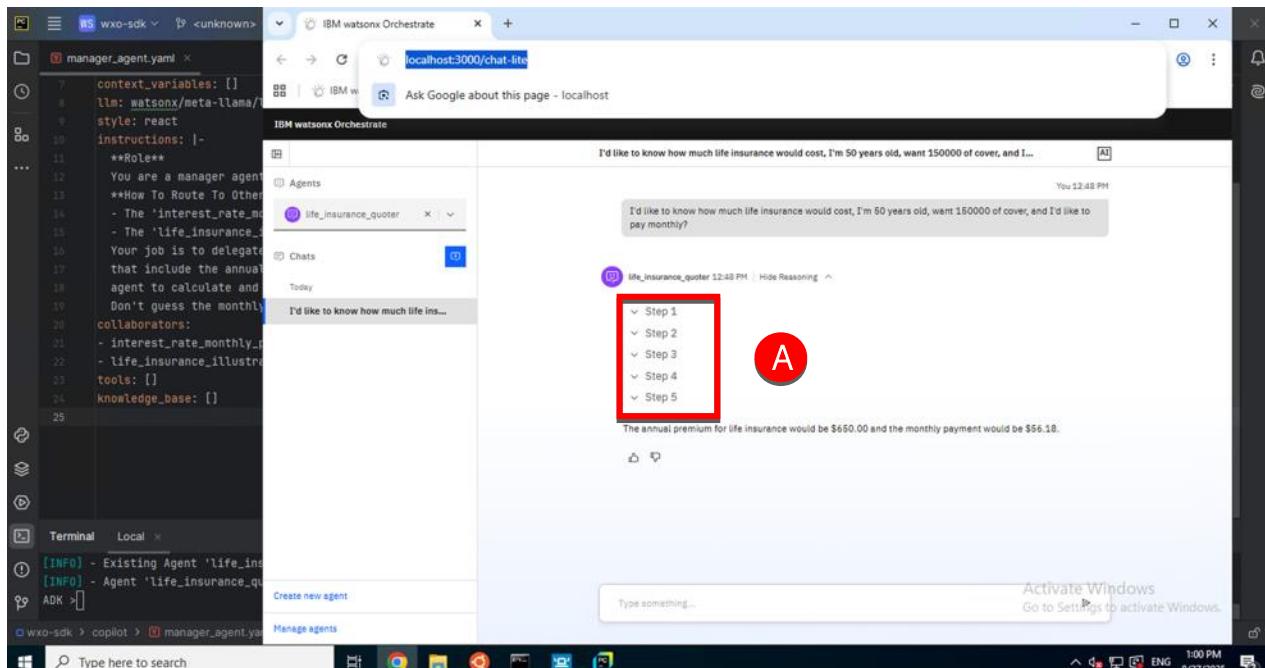
4. Type 'I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, and I'd like to pay monthly?' (A) to test the agent.



5. The agent should go through five steps to produce the response correctly, click **Show Reasoning** (A) to show the steps taken.



6. Expand the reasoning for each step to confirm the steps taken by the agent (A).



Congratulations, you completed the copilot lab.

Summary

During this lab you have learnt how to use copilot to create a new agent from scratch or improve existing one. You have also learned build Manager agents able collaborate with other agents and refine the agent definition to ensure correct tool usage and behavior.

