Lesson 4 Demo 01
Build the Right Tools

Objective: To demonstrate building essential tool files that empower AI agents in a CrewAI-based trip planner

These tools enhance the agent's ability to retrieve real-time travel data and perform necessary calculations, ensuring accurate and efficient itinerary planning. The following tools will be developed to serve two key purposes:

1. Search tool: To fetch real-time travel information from external sources
2. Calculator tool: To perform essential computations for itinerary planning

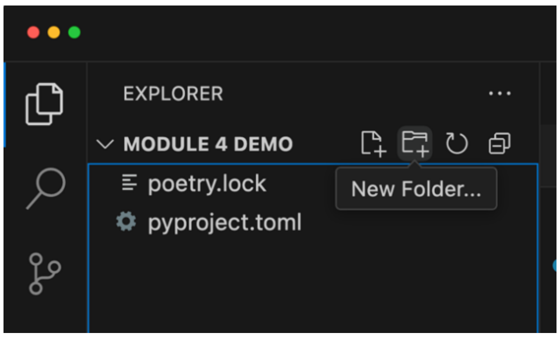
Tools required: VSCode

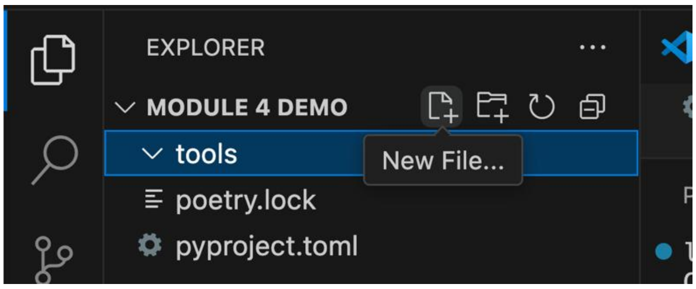
Prerequisites: Complete the lesson 4 prerequisite demo

Steps to be followed:

1. Create a new folder named *tools* inside the project folder
2. Create a `calculator_tools.py` inside this folder
3. Create a `search_tools.py` inside the same folder
4. Save the files

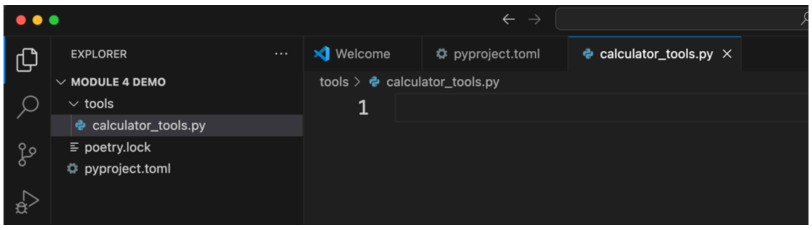
Step 1: Create a new folder named *tools* inside the project folder



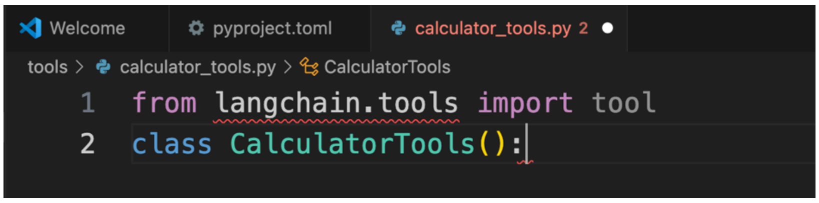


Step 2: Create a *calculator_tools.py* inside this folder

2.1 Create the new file `calculator_tools.py`



2.2 Define the class: Create `CalculatorTools`, which provides a tool for mathematical calculations.

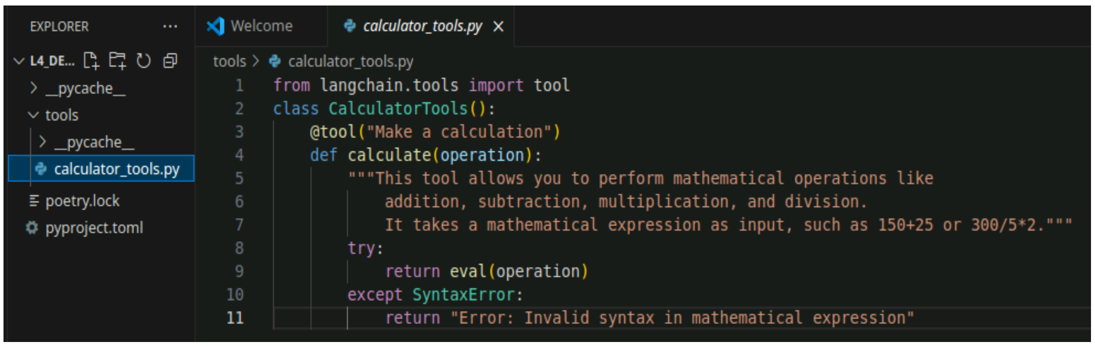


```
from langchain.tools import tool
class CalculatorTools():
```

2.3 Register the tool: Use the `@tool("Make a calculation")` decorator to make the calculate method available for execution.

2.4 Evaluate expressions: Accept a mathematical expression as a string and compute the result using `eval()`.

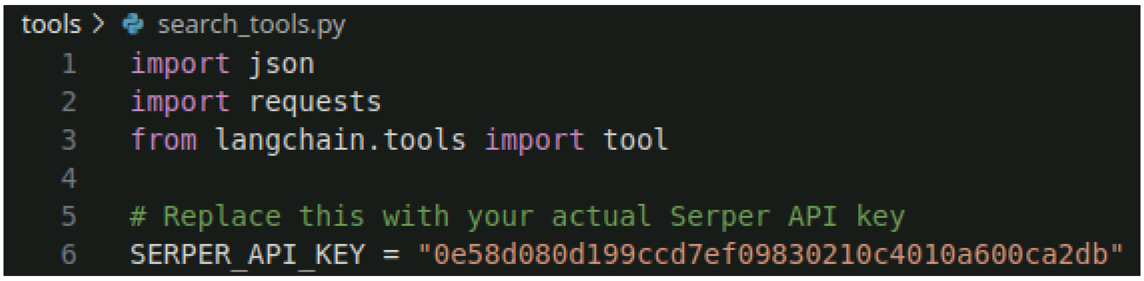
2.5 Handle errors: Catch syntax errors and return a friendly error message if the input is invalid.



```python
from langchain.tools import tool
class CalculatorTools():
    @tool("Make a calculation")
    def calculate(operation):
        """This tool allows you to perform mathematical operations like
            addition, subtraction, multiplication, and division.
            It takes a mathematical expression as input, such as 150+25 or 300/5*2."""
        try:
            return eval(operation)
        except SyntaxError:
            return "Error: Invalid syntax in mathematical expression"
```

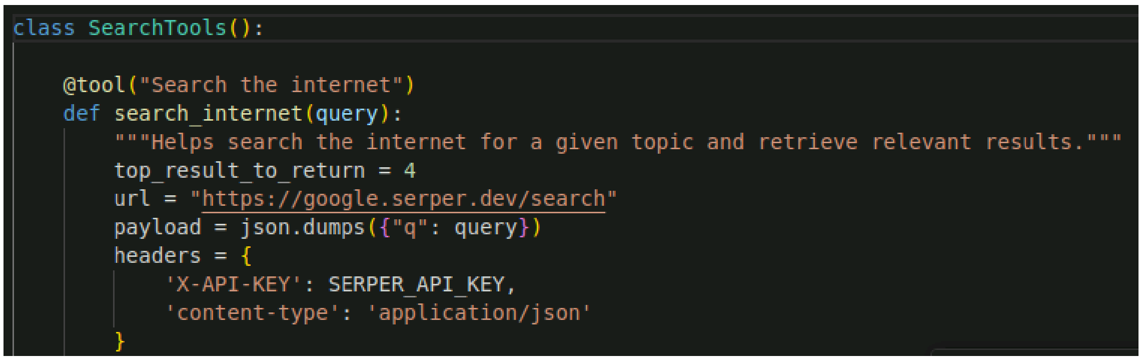
Step 3: Create a *search_tools.py* inside the same folder

3.1 Import dependencies: Load necessary modules (`json, requests`) and the tool decorator from `langchain.tools,` also create the variable to store SERPER API KEY.



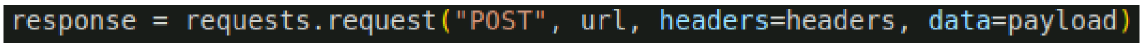
```python
import json
import requests
from langchain.tools import tool

# Replace this with your actual Serper API key
SERPER_API_KEY = "0e58d080d199ccd7ef09830210c4010a600ca2db"
```

3.2 Define the class: Create `SearchTools` with a method `search_internet`, registered as a tool for internet searches.



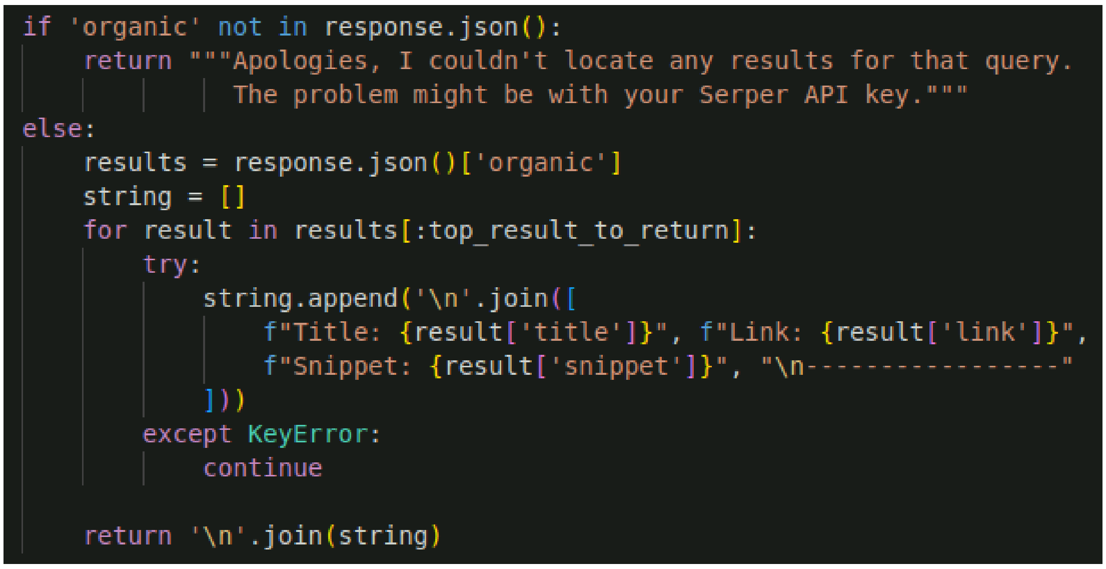
```python
class SearchTools():

    @tool("Search the internet")
    def search_internet(query):
        """Helps search the internet for a given topic and retrieve relevant results."""
        top_result_to_return = 4
        url = "https://google.serper.dev/search"
        payload = json.dumps({"q": query})
        headers = {
            'X-API-KEY': SERPER_API_KEY,
            'content-type': 'application/json'
        }
```

3.3 Make an API request: Send a `POST` request to `Serper API` with the search query, using an API key from environment variables.



```python
response = requests.request("POST", url, headers=headers, data=payload)
```

3.4 Process and return results: Extract and format the top search results, handling errors if no results are found



```python
if 'organic' not in response.json():
    return """Apologies, I couldn't locate any results for that query.
            The problem might be with your Serper API key."""
else:
    results = response.json()['organic']
    string = []
    for result in results[:top_result_to_return]:
        try:
            string.append('\n'.join([
                f"Title: {result['title']}", f"Link: {result['link']}",
                f"Snippet: {result['snippet']}", "\n-----------------"
            ]))
        except KeyError:
            continue

    return '\n'.join(string)
```

Step 4: Save the files