AI Engineering Task: LangChain Tools Agent

Reference: LangChain Tools (Python)

Objective

Build a terminal-based AI agent using **LangChain Tools** for real-time tool calling (web search, math, and custom utilities) with lightweight conversational memory.

Task Description

Create a working prototype that:

- Runs locally from the terminal using a `while True` loop.
- Uses **LangChain Tools** with an OpenRouter LLM endpoint.
- Integrates **Tavily Search API** for internet search.
- Maintains short-term memory and displays tool invocation traces.

Core Features

1. **Tool Registry**
2. Internet Search (via Tavily API)
3. Math Evaluator
4. Custom tools
5. **Agent Composition**
6. LLM connected to tools through LangChain.
7. In-memory conversational history.
8. Structured tool call logs printed to the terminal.
9. **Main Execution Flow**
10. Implemented in a single `while True` loop within `main.py`.
11. Handles user input, tool calls, and conversation flow.

File Structure

```
project_root/
│
├── tools.py        # Contains all LangChain tool definitions
├── agent.py        # Defines the LLM agent, memory, and bindings
├── main.py         # Entry point with while True chat loop
├── .env.example    # Environment variables (API keys placeholders)
└── README.md       # Setup, usage, and run instructions
```

Implementation Details

1. Environment Setup

- **Python:** 3.13+
- **Install Packages:**

```
pip install langchain langchain-core langchain-community tavily-python
openai python-dotenv rich
```

- **Environment Variables (.env):**

```
OPENROUTER_API_KEY="your_openrouter_key"
TAVILY_API_KEY="your_tavily_key"
```

2. Tools (`tools.py`)

- **Search Tool:** Wraps Tavily search with `top_k` and safe defaults.
- **Math Tool:** Safe mathematical expression evaluator.
- **Custom Tool**

Follow LangChain's tool interface and schema for defining input/output.

3. Agent (`agent.py`)

- Instantiate the LLM (via OpenRouter endpoint).
- Bind tools with LangChain's binding utilities.
- Integrate lightweight in-memory message history.
- Add clear print logs for tool invocations and outputs.

4. Main File (`main.py`)

- Import the agent from `agent.py`.
- Start a continuous input loop:

```
while True:
    user_input = input("You: ")
    if user_input.lower() in ["exit", "quit"]:
        break
    response = agent.run(user_input)
    print(f"Agent: {response}")
```

Deliverables

- Functional prototype with three files:
- `tools.py` (Tool definitions)
- `agent.py` (Agent + bindings)
- `main.py` (Execution loop)
- `.env.example` for API key setup
- `README.md` with installation and usage instructions

Evaluation

- Agent runs in terminal and responds interactively.
- Tools correctly register and invoke.
- Tavily search and math tool return deterministic results.
- Custom `ticker_info` responds with mock data.
- Conversation memory maintains previous context.

Notes

- No backend, UI, or external DB is needed.
- The focus is on understanding LangChain Tool registration, invocation, and conversational chaining.
- Ensure clean, minimal, and reproducible implementation.
