```
#!/usr/bin/env python

import argparse
import json
import xml.etree.ElementTree as ET

import requests

URL_OLLAMA = "http://localhost:11434/api/chat"
URL_MCP_SERVER = "http://localhost:5000/mcp"


def get_system_prompt(tool_name, tool_desc, tool_input):
    SYSTEM_PROMPT = f"""You are an AI quiz assistant.
Please present a quiz to the user and evaluate their input as either correct or incorrect.
The correct answer to the quiz must be one of the following: dog, cat, horse, or giraffe.

## Message Format

Your response must follow the XML format shown below.
Only two tags are allowed within the <response> tag: <message> and <call>.
The <message> tag is mandatory, while the <call> tag is optional.

Follow this format strictly and do not add any extra lines.
Do not wrap the reply message in a Markdown code block.
If the user's answer is incorrect, provide additional hints and continue the quiz until the correct answer is given.

## Example Response

Replace %TOOL_NAME%, %PARAM_NAME%, and %PARAM_VALUE% according to the MCP Tools section.

<response>
<message>Correct! Cool answer!</message>
<call><name>TOOL_NAME</name><arguments><%PARAM_NAME%>%PARAM_VALUE%</%PARAM_NAME%></arguments></call>
</response>

Sample <call> Tag

<call><name>add</name><arguments><lhs>1</lhs><rhs>2</rhs></arguments></call>

## MCP Tools

You can use the following MCP Tools.

### TOOL_NAME: {tool_name}

Description:
{tool_desc}

Input Schema:
{tool_input}
"""

    return SYSTEM_PROMPT

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", default="gemma3:12b")
    args = parser.parse_args()
    return args


def get_response(model, messages):
    data = {
        "model": model,
        "messages": messages,
        "stream": False,
    }
    response = requests.post(URL_OLLAMA, json=data)
    if not response.ok:
        print(f"request failed due to {response.status_code}")
        raise Exception("invalid response")

    return response.json()["message"]


def init_mcp_server(mcp_session, url):
    """
    Init MCP server and get its tools.

    Continue MCP session with id = 2.

    Parameters
    ----------
    mcp_session: requests.Session
    url: MCP server URL

    Returns
```

```
    -------
    Tools List Response
      See https://modelcontextprotocol.io/docs/learn/architecture
      Tools Discovery > Tools List Response
    """
    mcp_session.headers.update({
        "Content-Type": "application/json",
        "Accept": "application/json, text/event-stream",
    })

    # initialization
    mcp_init_msg = {
        "jsonrpc": "2.0",
        "id": 1,
        "method": "initialize",
        "params": {
            "protocolVersion": "2025-06-18",
            "capabilities": {
                "elicitation": {}
            },
            "clientInfo": {
                "name": "example-client",
                "version": "1.0.0"
            }
        }
    }
    res0 = mcp_session.post(url, json=mcp_init_msg, stream=True)
    if not res0.ok:
        raise Exception("fail to initialize MCP server")

    mcp_session.headers.update(
        {
            "mcp-session-id": res0.headers["mcp-session-id"]
        }
    )

    # notification
    mcp_notification_msg = {
        "jsonrpc": "2.0",
        "method": "notifications/initialized",
    }
    res1 = mcp_session.post(url, json=mcp_notification_msg, stream=True)
    if not res1.ok:
        raise Exception("notifications/initialized failed")

    # get MCP tools
    mcp_list_tool_msg = {
        "jsonrpc": "2.0",
        "id": 2,
        "method": "tools/list",
        "params": {
            "cursor": "optional-cursor-value",
        },
    }
    res2 = mcp_session.post(
        URL_MCP_SERVER,
        json=mcp_list_tool_msg,
    )
    if not res2.ok:
        raise Exception("tools/list failed")

    return res2


def call_mcp_tool(mcp_session, url, mcp_id, tool_name, tool_args):
    """
    Call MCP tool.

    Parameters
    ----------
    mcp_session: requests.Session
    url: MCP server URL
    mcp_id: mcp message id
    tool_name: tool name
    tool_args: xml.etree.ElementTree.Element
    """
    params = {
        "name": tool_name,
        "arguments": {c.tag: c.text for c in tool_args},
    }

    body = {
        "jsonrpc": "2.0",
        "id": mcp_id,
        "method": "tools/call",
        "params": params,
    }

```

```
    # print(f"call MCP tool: {body=}")

    res = mcp_session.post(url,
                           json=body)
    if not res.ok:
        raise Exception("tools/call failed")


def main():
    args = parse_args()

    # init MCP server
    mcp_session = requests.Session()
    res2 = init_mcp_server(mcp_session, URL_MCP_SERVER)

    print("---- res2 -----")
    print(res2.text)
    print("---- res2 -----")
    print("")

    tools_json = json.loads(res2.text.split("¥r¥n")[1][6:])

    # get first tools
    assert len(tools_json["result"]["tools"]) == 1
    tool = tools_json["result"]["tools"][0]

    tool_name = tool["name"]
    tool_desc = tool["description"]
    tool_input = tool["inputSchema"]

    # send system prompt to get first quiz
    messages = [
        {
            "role": "system",
            "content": get_system_prompt(tool_name, tool_desc, tool_input)
        }
    ]

    res_msg = get_response(args.model, messages)
    # Uncomment the line below to display the first quiz
    print(res_msg)
    messages.append(res_msg)
    assistant_msg = res_msg["content"]

    mcp_id = 2
    while True:
        root = ET.fromstring(assistant_msg)
        msg = root.find("message").text

        # call MCP tool
        call = root.find("call")
        if call is not None:
            tool_name = call.find("name").text
            tool_args = call.find("arguments")

            call_mcp_tool(mcp_session, URL_MCP_SERVER,
                          mcp_id, tool_name, tool_args)
            mcp_id += 1

        # show chat message
        print(msg)
        user_input = input("input answer: ")
        user_input.strip()

        # append user message
        messages.append(
            {
                "role": "user",
                "content": user_input,
            }
        )

        res_msg = get_response(args.model, messages)
        # Uncomment the line below to display the raw response
        # print(res_msg)
        assistant_msg = res_msg["content"]

        # append agent message
        messages.append(res_msg)


if __name__ == "__main__":
    main()
```