Review of `mcp_server.py`

1. Syntax and Formatting

1. **Indentation of `raise RuntimeError`**
   The `raise RuntimeError("Set OPENWE_API_KEY environment variable")` is not indented under the `if not API_KEY:` block. This will cause an `IndentationError`.
2. **Missing parenthesis in `httpx.get` call**
   In the `get_weather_update` function, the call to `client.get` has an unmatched parenthesis due to a trailing comma.
3. **PEP8 Line Length**
   Some lines exceed 79 characters (e.g., the URL constant and long docstrings).

2. Logical Issues

1. **No Timeout for HTTP Request**
   The `httpx.AsyncClient().get(...)` call does not specify a timeout, which may hang indefinitely.
2. **Returning Raw JSON vs. Structured Response**
   The `get_weather_update` returns the entire API JSON. It may be preferable to wrap it or validate its structure.
3. **Unit Consistency**
   The weather tool always returns metric units (“metric”). If users need imperial, there is no option.

3. Suggested Improvements

- Add timeout and error handling for network issues.
- Properly indent the runtime error raise.
- Fix the unmatched parenthesis.
- Optionally parameterize units.
- Add docstrings and type hints for FastAPI routes.

Review of `multi_agent.py`

1. Syntax and Formatting

1. **Missing Commas in Config Constructors**
   - In `AzureAIAgentConfig(...)`, there is no closing parenthesis and comma placement is inconsistent.
   - In `AzureAIAgent(...)` constructors for `math_agent`, `weather_agent`, and `supervisor_agent`, missing commas and mismatched parentheses.
2. **Unclosed Strings**
   - The `math_system_instruction` and `weather_system_instruction` strings are not properly closed with parentheses.
3. **Incorrect `run_async_with_intent` Usage**
   - The call to `run_async_with_intent` passes `query, agents=[...]` but likely requires named parameters, e.g., `prompt=query, agents=[...]`.

2. Logical Issues

1. **Supervisor Delegation Logic**
   - The `if "weather" in query.lower():` check is too brittle. Better to delegate via intent detection.
2. **Missing Skill Import for Supervisor**
   - `kernel.import_plugin` only imports `mcp`; supervisor has no tools; OK, but consider passing empty list.
3. **Error Handling**
   - No exception handling for network or agent failures.

3. Suggested Improvements

- Correct parentheses and commas.
- Close strings properly.
- Use explicit parameter names for `run_async_with_intent`.
- Enhance intent detection with a more flexible approach.
- Wrap agent runs in `try/except` to catch errors.

Updated Code

`mcp_server.py` (Corrected)

```
import os
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import httpx
from fastmcp import FastMCP
```

```python
# Load API key from environment
API_KEY = os.getenv("OPENWEATHER_API_KEY")
if not API_KEY:
    raise RuntimeError("Set OPENWEATHER_API_KEY environment variable")

WEATHER_API_URL = "https://api.openweathermap.org/data/2.5/weather"

# Initialize FastAPI & FastMCP
app = FastAPI(title="Calculator & Weather MCP Server")
mcp = FastMCP("CalculatorWeatherMCP")

class CalcParams(BaseModel):
    a: float
    b: float

class WeatherParams(BaseModel):
    city: str
    units: str = "metric"

@mcp.tool()
def add_number(params: CalcParams) -> dict:
    """Add two numbers."""
    return {"result": params.a + params.b}

@mcp.tool()
def multiply_number(params: CalcParams) -> dict:
    """Multiply two numbers."""
    return {"result": params.a * params.b}

@mcp.tool()
async def get_weather_update(params: WeatherParams) -> dict:
    """Retrieve current weather for a city."""
    async with httpx.AsyncClient(timeout=10.0) as client:
        resp = await client.get(
            WEATHER_API_URL,
            params={"q": params.city, "appid": API_KEY, "units": params.units},
        )
    if resp.status_code != 200:
        raise HTTPException(status_code=resp.status_code, detail=resp.text)
    return resp.json()

# Expose tools via FastAPI under /mcp
mcp.from_fastapi(app, path="/mcp")

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

`multi_agent.py` (Corrected)

```python
import os
import asyncio
from semantic_kernel import Kernel
from semantic_kernel.ai_azure import AzureAIAgentConfig, AzureAIAgent
from semantic_kernel.connectors.mcp import MCPHttpPlugin
```

```python
# Load Azure credentials
AZURE_OPENAI_ENDPOINT = os.getenv("AZURE_OPENAI_ENDPOINT")
AZURE_OPENAI_KEY = os.getenv("AZURE_OPENAI_KEY")
if not AZURE_OPENAI_ENDPOINT or not AZURE_OPENAI_KEY:
    raise RuntimeError("Set AZURE_OPENAI_ENDPOINT and AZURE_OPENAI_KEY environment variab

# MCP server HTTP-streaming endpoint
MCP_HTTP_URL = "http://localhost:8000/mcp"

async def main():
    # 1. Initialize the Semantic Kernel
    kernel = Kernel()

    # 2. Discover and import MCP tools
    async with MCPHttpPlugin(name="FastMCPTools", url=MCP_HTTP_URL) as mcp_plugin:
        kernel.import_plugin(mcp_plugin, skill_collection_name="mcp")

    # 3. Configure Azure AI agent settings
    agent_config = AzureAIAgentConfig(
        endpoint=AZURE_OPENAI_ENDPOINT,
        api_key=AZURE_OPENAI_KEY,
        model_deployment_name="gpt-4o"
    )

    # 4. Create specialist agents
    math_system_instruction = (
        "You are the Math Agent. Use only the provided tools add_number and "
        "multiply_number. When asked, call the appropriate tool with numeric "
        "inputs and return only the tool’s JSON response containing the 'result'."
    )
    math_agent = AzureAIAgent(
        config=agent_config,
        kernel=kernel,
        skills=["mcp.add_number", "mcp.multiply_number"],
        system_message=math_system_instruction
    )

    weather_system_instruction = (
        "You are the Weather Agent. Use only the provided tool get_weather_update. "
        "When asked for weather, call the tool with the city name and return only "
        "the JSON payload from the weather API. Do not add explanations."
    )
    weather_agent = AzureAIAgent(
        config=agent_config,
        kernel=kernel,
        skills=["mcp.get_weather_update"],
        system_message=weather_system_instruction
    )

    supervisor_system_instruction = (
        "You are the Supervisor Agent. Analyze the user's query and decide whether "
        "it is math-related or weather-related. Delegate the query to math_agent or "
        "weather_agent accordingly using run_async_with_intent. Return exactly the "
        "specialist agent's response."
    )
    supervisor_agent = AzureAIAgent(
        config=agent_config,
        kernel=kernel,
        skills=[],
        system_message=supervisor_system_instruction
    )

    # 5. Example usage
    sum_result = await math_agent.run_async(prompt="add_number", a=5, b=7)
    prod_result = await math_agent.run_async(prompt="multiply_number", a=3, b=4)
    weather = await weather_agent.run_async(prompt="get_weather_update", city="London")
    print("Sum:", sum_result)
    print("Product:", prod_result)
    print("Weather:", weather)

    # 6. Supervisor delegates based on intent
    query = "What’s the weather in Paris?"
    try:
        result = await supervisor_agent.run_async_with_intent(
            prompt=query,
            agents=[weather_agent, math_agent]
        )
        print("Supervisor Result:", result)
    except Exception as e:
        print("Error during supervision:", e)

if __name__ == "__main__":
    asyncio.run(main())
```

⁂