2b. Why Is `user_prompt` Passed in `Runner.run()`, and Why Is It a `@classmethod`?

🧠 Understanding the Concept

According to the OpenAI Agents SDK documentation, the `Runner.run()` method is defined as a `@classmethod` that takes both an `Agent` object and a `user_prompt` string as arguments.

But why is this necessary? Let's break it down step by step to understand **why the `user_prompt` must be passed** and **why `Runner.run()` is implemented as a `classmethod`.**

📥 Why Is the `user_prompt` Passed?

The `user_prompt` is the **specific instruction or question** provided by the user, such as:

- “What’s the weather in Karachi?”
- “Plan a trip to Paris.”

This input is dynamic and changes with each request. Here's why it's important to pass it into `Runner.run()`:

1. Dynamic Input

- Every time the user interacts with the agent, their question or command is different.
- If this prompt were hardcoded inside the `Agent` or `Runner` class, the agent would only be able to perform one task, losing flexibility.

2. Task-Specific Behavior

- The system prompt (set inside the `Agent.instructions`) defines the **agent’s role** (e.g., "You are a travel planner").
- The `user_prompt`, on the other hand, defines **what exact task** the agent should perform at that moment (e.g., "Check the weather in Karachi").

3. Flexibility and Reusability

- By passing the `user_prompt` into `Runner.run()`, we make the agent **reusable** for multiple different queries—without changing the code.

🗒️ **Analogy:**

Think of the `Runner` as a chef and the `user_prompt` as a customer's order. Every customer asks for a different dish—pizza, biryani, etc.—so the order must be passed to the chef every time. If the chef could only make one hardcoded dish, that wouldn't work in a real kitchen!

🧩 Why Is `Runner.run()` a Classmethod?

The `Runner.run()` method is implemented as a `@classmethod` so that it can be called **without creating an instance of `Runner`**. This design has several advantages:

1. Stateless Execution

- Since `Runner` doesn't need to store any internal state (like instance variables), we don't need to create a new object every time.
- This makes the code **simpler and more efficient**.

2. Simplified API

You can directly call:

Runner.run(agent, "What is the weather in Karachi?")

- —without creating an instance like `runner = Runner()`.

3. Code Reusability

- One `Runner` class can be used with multiple `Agent` objects and different prompts. There's no need to manage separate instances.

4. Easier Testing

- Since there's no internal state, unit tests can directly test `Runner.run()` by just passing inputs and checking outputs—no setup required.

🧾 **Analogy:**

Imagine `Runner` as a restaurant manager who coordinates orders between customers and chefs. You don't hire a new manager for each order—you just pass the order, and the manager handles it. That's how `classmethod` works here.

💻 Code Example

Here's a basic example showing how `Runner.run()` works:

```
from dataclasses import dataclass


@dataclass
class Agent:
    instructions: str


class Runner:
    @classmethod
    def run(cls, agent: Agent, user_prompt: str) -> str:
        print(f"Running with instructions: {agent.instructions}")
        return f"Response: {user_prompt}"


# Usage
agent = Agent(instructions="You are a helpful assistant")
response = Runner.run(agent, "What is the weather in Karachi?")
print(response)
```

Output:

```
Running with instructions: You are a helpful assistant
Response: What is the weather in Karachi?
```

🧪 Practice Code: Travel Planner Agent

Here's a more advanced example that simulates a travel planner agent using a mock SDK to return weather information:

```
from dataclasses import dataclass

from typing import List


# Simulated SDK tool

class SDK:

    def get_weather(self, city: str) -> str:

        return f"The weather in {city} is sunny with a temperature of 25°C."


@dataclass

class Agent:

    instructions: str

    tools: List[str]


class Runner:

    @classmethod

    def run(cls, agent: Agent, user_prompt: str, sdk: SDK) -> str:

        print(f"Running with tools: {agent.tools}")

        city = user_prompt.split()[-1] if "weather" in user_prompt.lower() else "unknown"

        if "weather" in user_prompt.lower():

            weather = sdk.get_weather(city)

            return f"{agent.instructions}: {user_prompt}\n{weather}"

        return f"{agent.instructions}: {user_prompt}"
```

```
# Usage

sdk = SDK()

agent = Agent(instructions="You are a travel planner", tools=["weather_api"])

response1 = Runner.run(agent, "What is the weather in Karachi?", sdk)

print(response1)

response2 = Runner.run(agent, "Plan a trip to Paris", sdk)

print(response2)
```

Author

Written by Humaiza naz