Task 07: OpenAI Agents SDK Research

🔗 Reference Links:

- OpenAI Agents First Example
- Agent Class Documentation
- Runner Class Documentation

---

1. Why is the Agent class defined as a `@dataclass`?

➤ Explanation:

Python's `@dataclass` decorator simplifies class creation when the class is mainly used to store data.

✅ Benefits in the context of `Agent`:

- Reduces boilerplate code (`__init__`, `__repr__`, etc.)
- Automatically generates useful methods.
- Makes it easy to read, maintain, and instantiate the `Agent` with required fields like tools, instructions, etc.

🧪 Example:

```python
from dataclasses import dataclass

@dataclass
class Agent:
    name: str
    instructions: str
    tools: list
```

2a. Why are `instructions` (system prompts) part of the Agent class? Why can it also be callable?

➤ Explanation:

The `instructions` field acts as the **system message** — it defines the agent's identity and behavior.

But `instructions` can also be passed as a **callable function**, allowing the agent to **dynamically generate its prompt** based on context.

□ Example:

```
# Static instructions
Agent(instructions="You are a helpful assistant.")

# Dynamic/callable instructions
def generate_instructions(context):
    return f"You are helping with task: {context['task_name']}"

Agent(instructions=generate_instructions)
```

This gives flexibility for building context-aware agents.

2b. Why is the user prompt passed in the `run()` method of `Runner` class? And why is it a `@classmethod`?

➤ Explanation:

The `run()` method is responsible for executing the agent's logic when the user sends a message.

- The **user prompt** is not stored in the agent. Instead, it is passed during execution via the `run()` method.
- `run()` being a `@classmethod` allows it to be called on the class itself rather than on an instance. This is useful for running agents statelessly or within a specific context setup.

□ Example:

```
response = Runner.run(agent, input="What's the weather today?")
```

---

3. What is the purpose of the `Runner` class?

➤ Explanation:

The `Runner` class handles the **execution flow** of the agent. It does things like:

- Receiving the user's input
- Calling the agent's logic
- Managing context and tools
- Returning the final response

So basically, the `Agent` is the brain, and the `Runner` is the one that activates it.

---

4. What are Generics in Python? Why is `TContext` used as a Generic?

➤ Explanation:

Generics are a feature that lets you write **type-safe** and **reusable** code by using type variables. In Python, you use them with `TypeVar`.

`TContext` is a placeholder for any context type passed around in agents or runners.

This is especially useful when you want to write code that works for **any type of context**, like a dictionary, object, or custom class.

□ Example:

```
from typing import TypeVar, Generic

TContext = TypeVar('TContext')

class MyAgent(Generic[TContext]):
    def __init__(self, context: TContext):
        self.context = context
```

✓ Summary:

Concept	Purpose
`@dataclass`	Clean, minimal class for holding agent data
`instructions`	Defines system prompt (can be static or dynamic)

Concept	Purpose
`run()`	Executes agent logic with user input
`Runner`	Manages execution flow of the agent
`Generics (TContext)`	Ensures flexible & type-safe context usage