**Q1: The Agent class has been defined as a dataclass. Why?**

A **dataclass** in Python is used to automatically generate boilerplate code like \_\_init\_\_, \_\_repr\_\_, and comparison methods.  
The Agent class is a dataclass because:

* It mostly stores **configuration and data** (like name, instructions, tools, etc.).
* Using @dataclass makes the code **cleaner and more readable**.
* It helps avoid writing long constructors and makes it easy to **instantiate agents with specific parameters**.

**Q2a: The system prompt is contained in the Agent class as instructions. Why can you also set it as a callable?**

Yes, instructions in the Agent class can be a **string or a callable** (like a function). This is useful because:

* If instructions is a **string**, the system prompt is **static**.
* If it's a **callable**, the system prompt can be **dynamic**, meaning:
  + It can change based on the **context** or other inputs.
  + This makes the agent more **flexible and smart**, depending on the situation.

So, setting instructions as a callable gives more power to customize agent behavior.

**Q2b: The user prompt is passed as a parameter in the run method of Runner, and the method is a classmethod. Why?**

The run method of Runner takes the user prompt because:

* It's the actual **user input** that needs a response.
* Since the agent (system prompt) is already defined in Agent, the user's message comes in when we **execute or run** the agent.

The method is a **@classmethod** because:

* It does not depend on an **instance of Runner**, but rather works at the **class level**.
* It can **set up the context and start the agent task** without needing to create a Runner object manually.

So it's designed this way for simplicity and better structure.

**Q3: What is the purpose of the Runner class?**

The Runner class handles the **execution** of the agent’s task. Its main purposes are:

* To **run the agent** using the prompt and tools.
* To manage the **context**, **tool use**, and **final output**.
* It acts like the **engine** behind the agent’s interaction with the user.

Think of it like this:

* Agent = Definition
* Runner = Execution

**Q4: What are generics in Python? Why do we use it for TContext?**

Generics in Python (like TContext) allow us to:

* Write code that works with **any type**, but still has **type hints**.
* For example, TContext is a **type variable**, so we can define a context that is **customized** to our agent.

We use TContext in the SDK so:

* Each agent can have its **own context structure**.
* The SDK stays **flexible and reusable** for different types of agents.

**✅ Summary**

* Agent is a dataclass to store config easily.
* instructions can be dynamic using callables.
* User prompt goes into Runner.run() because it's the entry point of execution.
* Runner runs the task; it's like the "worker."
* TContext is a generic for flexible, typed context across agents.