🧠 What Are `instructions` in the `Agent` Class?

In the OpenAI **Agents SDK**, the `instructions` field inside the `Agent` class defines the **system prompt** — i.e., the initial guide that tells the agent **how to behave**.

✅ Think of `instructions` as:

> “What kind of assistant am I supposed to be?”

🔤 Example:

```
agent = Agent(
    name="Friendly Helper",
    instructions="You are a friendly assistant who helps users with
technical topics.",
    model="gpt-4"
)
```

- This string is sent as the **system message** in the chat.
- It shapes the agent’s **tone, role, and response strategy**.

💡 Real-Life Analogy:

Imagine giving a waiter a **note** saying:

- "Be super polite, recommend vegetarian dishes, and speak softly."

That’s like giving `instructions` to an AI agent:

- “Be helpful, creative, and avoid controversial topics.”

❓ So Why Can `instructions` Also Be a Callable?

This is where the real power comes in.

➕ Sometimes, you don’t want the same prompt every time.

You want the agent to:

- Customize its role **based on the user**
- Adapt to **current context or memory**
- Generate instructions **dynamically**

✅ Example: Callable Instructions

```
from dataclasses import dataclass

@dataclass
class UserContext:
    name: str
    goal: str

def dynamic_instructions(context: UserContext) -> str:
    return f"You are a personal assistant for {context.name},
helping them achieve: {context.goal}."

agent = Agent[UserContext](
    name="Dynamic Agent",
    instructions=dynamic_instructions,
    model="gpt-4"
)

ctx = UserContext(name="Hafsa", goal="master generics in Python")
print(agent.instructions(ctx))
```

🔽 Output:

```
You are a personal assistant for Hafsa, helping them achieve: master
generics in Python.
```

That’s **context-aware prompting** using `instructions` as a callable ✅

🧭 Summary: Why Instructions Can Be a Callable

Reason	Explanation
🧠 Dynamic Behavior	Adapts system message based on user/context/memory
🎯 Personalization	Tailors the agent’s role to the situation or person
🔄 Flexibility	One agent class can serve many types of users
⚙️ Reusability	Reuse the agent logic while customizing only the prompt
✅ Type Safety	With generics like `TContext`, the callable is still type-checked