Ottima domanda! Il motivo per cui **è necessario usare** `asyncio` (cioè la programmazione asincrona in Python) quando utilizzi gli **Agents dell'SDK di OpenAI** è principalmente legato a **come funzionano gli agenti** e a **come comunicano con i modelli e le tools**.

💡 Ecco perché è necessario `asyncio` con gli Agent:

1. **Interazione continua e asincrona con i modelli**
   Gli agenti non fanno solo una chiamata `chat()` e basta. Loro:

   - gestiscono step multipli (pensano, agiscono, osservano)
   - chiamano strumenti esterni (tools, function calling)
   - aspettano risposte da API o da altri task lenti

   Questo processo richiede **attendere eventi esterni**, e `asyncio` serve proprio a gestire queste attese in modo efficiente senza bloccare tutto il programma.

2. **Tools personalizzati spesso sono async** Se definisci tool personalizzati per l'agente, tipo: `python async def cerca_web(query: str) -> str: ...` ... allora l'agente stesso deve essere in grado di **chiamare funzioni async**, e per farlo **deve essere eseguito in un contesto asincrono**.

3. **Efficienza** Gli agenti possono eseguire operazioni I/O bound (API, file, rete...), e l'uso di `async` permette di **non bloccare il thread principale**. Quindi puoi avere un'app (tipo web app FastAPI, o bot Discord) che continua a funzionare mentre l'agente ragiona.

✅ Esempio pratico

```
import asyncio
from openai import AsyncOpenAI
from openai.agents import AsyncRunnableAgent

# Funzione tool asincrona
async def somma(a: int, b: int) -> int:
    return a + b

# Istanzia client e agent
client = AsyncOpenAI()
agent = AsyncRunnableAgent.from_tools([somma], client=client)

# Funzione principale asincrona
async def main():
    response = await agent.run("Qual è la somma di 4 e 7?")
    print(response)

# Avvio
asyncio.run(main())
```

Se invece cercassi di usare `run()` senza `await` e senza contesto `async`, Python ti darebbe errore tipo:
`RuntimeWarning: coroutine 'AsyncRunnableAgent.run' was never awaited`

Riassunto:

- Gli agenti OpenAI sono progettati per eseguire operazioni asincrone.
- `asyncio` serve per gestire attese non bloccanti, specialmente con tools, API e interazioni complesse.
- La versione `AsyncRunnableAgent` è specifica per l'ambiente `async`.

Se vuoi ti posso far vedere anche come usare gli agenti **senza** `asyncio`, ma con la versione **sincrona** (`RunnableAgent`), anche se meno flessibile. Fammi sapere!

Perfetto! 😎 Vediamo **come usare un agente di OpenAI SDK senza** `asyncio`, cioè in modalità **sincrona**.

✅ Esempio con `RunnableAgent` (sincrono)

Ecco un esempio base con un **tool sincrono** e l'agente che lavora senza `async`:

```
from openai import OpenAI
from openai.agents import RunnableAgent
from openai.types.beta.assistant import ToolFunction

# Tool sincrono
def somma(a: int, b: int) -> int:
    return a + b

# Tool definito per l'agente
tools = [
    ToolFunction.from_function(somma)
]

# Istanzia il client e l'agente
client = OpenAI()  # versione sincrona
agent = RunnableAgent.from_tools(tools=tools, client=client)

# Esegui la richiesta all'agente
response = agent.run("Quanto fa 12 più 30?")
print(response)
```

🧠 Differenze chiave con la versione `async`

Caratteristica	`RunnableAgent` (sync)	`AsyncRunnableAgent` (async)
Facile da usare in script semplici	✅	⚠️ richiede `asyncio`
Funziona con tools `def` normali	✅	✅ anche con `async def`
Ideale per ambienti tipo Jupyter o CLI	✅	❌ (a meno che non usi `nest_asyncio`)
Supporta chiamate async	❌	✅

⚠️ Attenzione:

Se anche **solo uno dei tuoi tool è** `async def`, allora **devi usare** `AsyncRunnableAgent`, altrimenti l'agente non potrà eseguirlo.

Vuoi che ti mostri anche **un esempio misto con più tool**, o magari un uso in uno **script Flask o FastAPI**?