

1st Step:

UV Kya Hai?

UV ek **fast Python package manager** hai — jo Python projects ke liye dependencies install karta aur manage karta hai . yeh `pip`, `poetry`, aur `conda` ka alternative hai.

AI agent ya chatbot banana ky lye uv **helper** hai — jo:

- Dependencies install karti hai
- Project ready karti hai
- Code chalne laayak bana deti hai

"UV tool khud agent nahi banata, lekin agent banane ke liye jo saman chahiye hota hai — wo sab ek button se ready kar deta hai." 🤖

```
uv init --package hello
```

Ek naya Python project banao jiska naam **hello** ho — aur usmein `pyproject.toml` file ban jaaye.

Chainlit Kya Hota Hai?

"Chainlit Python AI code ko ek chatbot ke tarah browser mein chala kar dikhata hai." 💬💻

```
uv add chainlit
```

```
uv run chainlit hello
```

Src ky bahar 1 file banao `chatbot.py`

```
import chainlit as cl

@cl.on_message
async def main(message: cl.Message):
    await cl.Message(
        content=f"Received: {message.content}",
    ).send()
```

```
Uv run chainlit run chatbot.py -w
```

2nd Step:

```
uv add openai-agents
```

openai-agents **AI agents banane ka powerful framework** deta hai — jisme agents define kar sakti hein, unko tools de sakti hein, aur woh intelligent behavior dikhate hain.

Topic	Yaad Rakho
🔑 <code>uv add openai-agents</code>	Ye tumhare project mein agents banana enable karta hai
<code>Agent()</code>	Tumhara intelligent bot yahin se start hota hai
🔌 <code>Runner.run_sync</code>	User ka input send karta hai agent ko
🔄 <code>AsyncOpenAI / LitellmModel</code>	Gemini ya GPT ko connect karta hai backend mein

```
uv add dotenv
```

`uv add` ka matlab hai **package install karna**. **dotenv** ek external library hai jo `.env` file se environment variables ko Python mein **load** karne ka kaam karti hai.

```
from dotenv import load_dotenv
```

Is line ka matlab hai: `.env` file se data load karne wali **function** ko import karo.
Ye sirf function ko import karta hai, actual load abhi tak nahi hua.

```
import os
```

"Ye Python ka **built-in module** hai is sy hm environment variables ko **read** ya **set** kar sakti hai jaise `os.getenv('KEY')`."

```
load_dotenv() #✔ Function  
set_tracing_disabled(True)
```

Ye function `.env` file ke andar likhe hue **key=value** pairs ko read karta hai .taake usme likhi hui **GEMINI_API_KEY** ya aur secrets Python mein aa jaayein.

Fir ye pairs ko system ke **environment variables** ke form mein set karta hai, jise `os.getenv()` se access kiya ja sakta hai.

"Ye line agent ke debug messages ko off kar deti hai — taake output saaf dikhe."

Jab bhi hum program run karte hain, **computer ya code** kuch **andar ki internal info** ya **background mein chal rahi cheezen** screen pe dikhata hai — inhi ko **debug messages** kehte hain. (Agent run hone ke waqt **debug trace** off karta hai (clean output ke liye))

Dotenv Kya Karta Hai?

`uv add dotenv` tumhare project mein `.env` file se secrets (like API keys) read karne ka system lagata hai — taake code clean aur secure rahe. ✓

Ye sensitive cheezen safe rakhta hai, jaise:

- API keys (e.g., GEMINI_API_KEY)
- Secrets
- Tokens
- Project settings

Feature	Kaam
<code>load_dotenv()</code>	<code>.env</code> file ko read karta hai
<code>os.getenv()</code>	Variable ko Python mein use karne deta hai
🔒 Secure	Code mein secret print nahi hota — sirf env mein hota hai

```
from agents import Agent, Runner, OpenAIChatCompletionsModel, AsyncOpenAI,
set_tracing_disabled
```

★ `agents`

→ Ye `uv add openai-agents` se install kiya package hai.

★ `Agent`

→ Ye ek **class** hai jo tumhara chatbot (agent) banata hai.

★ `Runner`

→ Agent ko run karne aur response lene ke liye use hota hai.

★ `OpenAIChatCompletionsModel`

→ Ye class tumhare LLM model (Gemini/GPT) ko wrap karti hai.

★ `AsyncOpenAI`

→ Ye Gemini ya OpenAI se **asynchronous tarike se baat** karne ka client object banata hai.

★ `set_tracing_disabled`

→ Ye ek optional function hai jo debug trace ko off karta hai (clean output ke liye).

Roman Urdu:

"Is line mein hum agent banane, model set karne aur agent run karne wali sari cheezen import kar rahe hain."

```
provider = AsyncOpenAI(  
    api_key=os.getenv("GEMINI_API_KEY"),  
    base_url="https://generativelanguage.googleapis.com/v1beta/openai/"  
)
```

🔍 `1. provider = AsyncOpenAI(...)`: Yeh ek variable hai (name tu kuch bhi rakh sakti ho). Is variable mein **AsyncOpenAI** ka object store ho raha hai.

✓ **AsyncOpenAI** ek class hai

Jo agents SDK ke andar hoti hai jo Gemini / OpenAI API ko (jaise Gemini ya GPT) ko backend se connect karta hai

Jab `AsyncOpenAI(...)` likhti hai, tu: Us class ka ek naya object bana rahi hoti hai — jisko `provider` jese naam mein store karti ho, jo async (asynchronous) OpenAI API handle karta hai.

✓ **Yahan provider ek variable hai**

Jo `AsyncOpenAI()` class ka object banata hai.

<code>class Car: pass</code> <code>my_car = Car()</code>	<code>provider = AsyncOpenAI(...)</code>
---	--

→ Yahan `provider` ek object hai jo `AsyncOpenAI` class se bana hai.

→ Variable ka naam `provider` hai, lekin wo ek object ko hold karta hai.

★ `AsyncOpenAI(...)`

→ Ye Gemini ya OpenAI se connect hone ka object banata hai.

★ `api_key=os.getenv("GEMINI_API_KEY")`

→ Ye `.env` file se `GEMINI_API_KEY` uthata hai. **Fixed name:** ✓ Yes

★ `base_url=...`

→ Ye Gemini API ka address (URL) hai. **Fixed name:** ✓ Yes

`AsyncOpenAI` object **background mein, bina rukay**, Gemini API se baat karega.

□ Example: Jab tum `await` lagati ho, to code rukta nahi — ye hota hai async ka magic!

Roman Urdu:

"Ye part Gemini API se baat karne ka object banata hai, jisme API key aur URL diya gaya hai."

Variable	Kaam
provider	Ye ek object banata hai jo Gemini API ko call karta hai
api_key=os.getenv(...)	.env se API key fetch karta hai
base_url=...	Gemini API ka endpoint URL hota hai

```
model = OpenAIChatCompletionsModel(
    model="gemini-2.0-flash-exp",
    openai_client=provider,
)
```

★ model

→ Ye variable hai jisme tumhara LLM model store hai. Ye define karta hai **kis model se baat karni hai** (Gemini 2.0 Flash)

★ OpenAIChatCompletionsModel(...)

✓ Ye class `agents` SDK ke andar hoti hai.

Jo tumhare **Gemini / OpenAI** model ke sath kaam karta hai — especially "chat completion" (yaani sawal ka jawab dena).

"Tumhara AI model (jaise Gemini ya GPT) se baat karne ke liye ek bridge ya zariya tayar karna."

✓ Ye model ke config ko wrap karta hai:

"Wrap karna" ka matlab hai:

Kisi cheez ko andar lena aur uska kaam asaan kar dena.

Jaise ek lifafa letter ko **wrap** karta hai – taake letter safe aur organized rahe.

Waise hi `OpenAIChatCompletionsModel` tumhare Gemini model aur client ko wrap karta hai — taake use karna easy ho jaye ✓

🔗 `openai_client=provider` kya karta hai?

✓ Ye tumhare pehle se banaye gaye **AsyncOpenAI object** ko connect karta hai

Jo API call handle karega. **Gemini API ko call karne ke liye provider pass kiya gaya hai**

Agent ko yeh batata hai ke **kis model se baat karni hai**, aur uska format kya hoga.

```
agent1 = Agent(
    name="Assistant",
    instructions="you are helpful assistant that solves basic math problem.",
    model=model)
```

★ `agent1`

Ye variable hai jisme tumhara agent ka object rakha gaya hai. `Agent()` aik class constructor hai jisme tum naam, instructions, aur model deti ho.

★ `name="Assistant"`

Ye tumhare agent ka naam hai.

★ `instructions=...`

Ye batata hai ke agent kis tarah ka behavior karega.

★ `model=model`

Ye batata hai ke agent kis model se baat kare.

Roman Urdu:

"Ye line tumhara chatbot (agent) banati hai — jiska naam Assistant hai aur wo maths problem solve karta hai."

```
response = Runner.run_sync(  
    starting_agent=Agent1,  
    input="What is 5 + 10?",  
)
```

★ `response`

Ye variable hai jisme agent ka answer aayega. Ye object hai jo `Runner.run_sync(...)` ke return se milta hai.

Runner ek class hai jo `agents` SDK ke andar hoti hai. ★ **Kaam:**

Ye tumhare **AI Agent** ko **chalaata hai** (run karta hai)

◆ **Function:** `run_sync(...)`

Ye `Runner` class ka method hai jo tumhare agent ko **synchronous mode** mein chalata hai.

★ `Runner.run_sync(...)`

Agent ko ek query ke saath run karta hai aur result print karte hain.

★ `starting_agent=Agent1`

Ye batata hai kaunsa agent run hoga. Ye specify karta hai koun sa agent query handle karega

★ `input="..."`

Ye user ka sawal hai, jo agent solve karega

Roman Urdu:

"Ye line agent ko ek question ke sath run karti hai — aur agent ka jawab response mein save hota hai." `print(response.final_output)`

```
★ print(...)
```

`print()` ek built-in function hai jo terminal par text/output dikhata hai.

```
★ response.final_output
```

Agent ka final reply hota hai — jo user ko milta hai. **Ye ek attribute/property hai object ke andar.**

Concept	Kaam
<code>.env + load_dotenv()</code>	Secure API keys use karna
<code>AsyncOpenAI / LitellmModel</code>	Gemini ya GPT se connect karne ka method
<code>Agent()</code>	Agent banane ka main logic
<code>Runner.run_sync()</code>	Agent ko run karne ka simple tariqa

Ek Line Mein Summary

Tumne ek **chatbot agent** banaya jo Gemini model se baat karta hai, question ka answer deta hai, aur result terminal par print hota hai — sab kuch `.env` file ki madad se safe tarike se hota hai. ✓

3rd Step:

Ab chainlit or agents ko join krna hy.

```
uv venv
```

`venv` = **Virtual Environment**

`uv venv` agent project ke liye ek **naya bag aur apni chhoti duniya banata hai**. Us duniya mein sirf wo cheezein hoti hain jo agent ko chahiye. Jab tum `uv run agent` karti ho, to agent sirf **us naye bag (venv)** se cheezein uthata hai — system ke dusre software usme interfere nahi karte.

Main.py ki file main () parmeter ky ander **user_input** ya koi sab hi name dy skty ho.

⇒ Input last line main message aye ga or print ki jgha return aye ga

```
@cl.on_chat_start
async def chat_start():
    await cl.Message(content="Welcome Asma CHATGBT").send()
```

```
import chainlit as cl
from src.agent.main import agentss # <- function import karo
```

```
@cl.on_message
async def main(message: cl.Message):
    reply = await agentss(message.content)
    await cl.Message(content=reply).send()
```

➔ "Mujhe src/agent/main.py file ke andar se agentss naam ka function chahiye."

```
@cl.on_message
```

Ye ek **decorator** hai — jo chainlit ko batata hai:

"Jab user message bheje, to neech wala function chalao."

Yani agar koi user "Hello" likhe UI mein, to ye `main()` function chalega.

```
async def main(message: cl.Message):
```

➔ Ye ek **async function** hai — iska matlab hai ke ye **asynchronous** kaam karega (AI se response lene mein time lagta hai, to rukay bina kaam chalta rahe).

➔ `async def main(message: cl.Message):`
