**1st Step:**

**👩‍🏫 UV Kya Hai?**

**UV** ek **fast Python package manager** hai — jo Python projects ke liye dependencies install karta aur manage karta hai

Tum agar AI agent ya chatbot banana chahti ho, to uv tumhari **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 c 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):      # Our custom logic goes here...      # Send a fake response back to the user      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 tum agents define kar sakti ho, unko tools de sakti ho, aur woh intelligent behavior dikhate hain.

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

|  |
| --- |
| uv add dotenv |

## 🧠 Dotenv Kya Karta Hai?

### 🔐 Environment variables (.env file) ko Python mein load karta hai.

Ye sensitive cheezen safe rakhta hai, jaise:

* API keys (e.g., GEMINI\_API\_KEY)
* Secrets
* Tokens
* Project settings

| **Feature** | **Kaam** |
| --- | --- |
| load\_dotenv() | .env file ko read karta hai |
| os.getenv() | Variable ko Python mein use karne deta hai |
| 🔐 Secure | Code mein secret print nahi hota — sirf env mein hota hai |

## 🔚 Ek Line Mein Summary:

**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.** ✅

|  |
| --- |
| from dotenv import load\_dotenv |

� � dotenv ek library hai jo .env file se environment variables load karti hai. � � load\_dotenv() function isko load karne ke liye use hota hai. — taake hum secrets ko .env file se access kar sakein."

|  |
| --- |
| from agents import Agent, Runner, OpenAIChatCompletionsModel, AsyncOpenAI, set\_tracing\_disabled |

📌 agents  
→ Ye tumne 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."

|  |
| --- |
| import os |

"OS ka module hum use karte hain taake hum apni .env file se keys ko le sakein."

| **Line** | **Kaam** |
| --- | --- |
| from dotenv import load\_dotenv | .env file se API key load karne ke liye |
| from agents import ... | AI agent, runner, model, and LLM client import karta hai |
| import os | Python ka built-in module — env variable read karne ke liye |

|  |
| --- |
| load\_dotenv() #✅ **Function**  set\_tracing\_disabled(True) |

→ Ye function .env file ko read karta hai, taake usme likhi hui GEMINI\_API\_KEY ya aur secrets Python mein aa jaayein.

"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.

| **Function** | **Kaam** |
| --- | --- |
| load\_dotenv() | .env file se GEMINI\_API\_KEY jaise variables load karta hai |
| set\_tracing\_disabled(True) | Agent run hone ke waqt **debug trace** off karta hai (clean output ke liye) |

|  |
| --- |
| provider = AsyncOpenAI(  api\_key=os.getenv("GEMINI\_API\_KEY"),  base\_url="https://generativelanguage.googleapis.com/v1beta/openai/"  ) |

📌 provider  
→ Ye ek **variable** hai jisme ek object rakha gaya 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.

📌 base\_url=...  
→ Ye Gemini API ka address (URL) hai.

**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.

📌 OpenAIChatCompletionsModel(...)

* **Gemini ya GPT** ek **AI model** hai.
* Lekin wo **direct Python se baat nahi kar sakta.**
* Tumhe ek **model-wrapper** chahiye jo tumhare Python code ko Gemini se mila sake.

Isliye hum banate hain:  
→ Ye tumhara Gemini model ka wrapper hai jo provider ke through API se baat karega.

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

**Roman Urdu:**  
"Ye line model define karti hai — isme bataya gaya ke kaunsa Gemini model use hoga."

| **Variable** | **Kaam** |
| --- | --- |
| model | Ye define karta hai **kis model se baat karni hai** (Gemini 2.0 Flash) |
| openai\_client=provider | Gemini API ko call karne ke liye provider pass kiya gaya hai |

|  |
| --- |
| 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.

📌 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."

| **Field** | **Meaning** |
| --- | --- |
| name="Assistant" | Agent ka naam (jo UI ya logs mein dikh sakta hai) |
| instructions=... | Agent ko ye bataya ja raha hai ke woh **math helper** ban ke behave kare |
| model=model | Is agent ke peeche Gemini LLM model use ho raha hai |

|  |
| --- |
| response = Runner.run\_sync(  starting\_agent=Agent1,  input="What is 5 + 10?",  ) |

📌 response  
→ Ye variable hai jisme agent ka answer aayega.

📌 Runner.run\_sync(...)  
→ se assistant ko run karte hain aur result print karte hain.

📌 starting\_agent=Agent1  
→ Ye batata hai kaunsa agent run hoga.

📌 input="..."  
→ Ye user ka sawal hota hai.

**Roman Urdu:**  
"Ye line agent ko ek question ke sath run karti hai — aur agent ka jawab response mein save hota hai."

| **Function** | **Kaam** |
| --- | --- |
| Runner.run\_sync(...) | Agent ko ek query ke saath run karta hai (sync mode mein) |
| starting\_agent=... | Ye specify karta hai koun sa agent query handle karega |
| input="..." | Ye user ka sawal hai, jo agent solve karega |

print(response.final\_output)

📌 print(...)  
→ Python ka built-in function hai.

📌 response.final\_output  
→ Agent ka final reply hota hai — jo user ko milta hai.

**Roman Urdu:**  
"Ye line agent ka answer print karti hai."

| **Function** | **Kaam** |
| --- | --- |
| response.final\_output | Agent ka final jawab (LLM ne kya reply diya) print karta hai |

| **Concept** | **Kaam** |
| --- | --- |
| .env + load\_dotenv() | Secure API keys use karna |
| AsyncOpenAI / LitellmModel | Gemini ya GPT se connect karne ka method |
| Agent() | Agent banane ka main logic |
| Runner.run\_sync() | 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 chninlit 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 likhy gey

* 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 neeche 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):