Week 2 → OpenAI Agents SDK

Asynchronous python, AsyncIO.

→ AsyncIO is a way to write a python code, it is the light version of multithreading.

[Multithreading → when two or more processes are executed concurrently on multiple threads]

→ AsyncIO was introduced in python 3.5, is a very lightweight of doing multithreading and doesn't actually involves threads at an operating system level.

→ It doesn't involves multiprocessing, where all python processor ~~put~~ spawn together and they all run together.

→ when ~~you~~ using multi-agent frameworks, when you have multiple threads hitting different api's.

→ In python, asyncio is implemented by using the asyncio library, which enables writing concurrent coding using async and await.

→ The core of asyncio, responsible for scheduling and executing coroutines and handling I/O events. It manages the flow of control b/w different coroutines.

async → Used for defining an asynchronous fn (a coroutine)

await → Used inside an async function to pause its execution until an "awaitable" object (like a coroutine or task) completes. This yield control back to the event loop allowing others task to run.

Planner agent, it will [illegible]
coming up with number of searches.

```
② async def do-some-processing():
      # Do some work
      return "Done!".
   result = await do-some-processing().
```

→ long version ~~decision~~ of asyncio.

⊙ Async provides a lightweight alternative to threading or multiprocessing

⊙ Function defined with async def are called coroutines - they're special functions that can be paused and resumed.

⊙ Calling a coroutine doesn't execute it immediately - it return a coroutine object

⊙ To actually run a coroutine you must await it, which schedules it for execution within an event loop ↓

→ ~~Spt~~ scheduled to executions A piece of code return with system library has a ~~[illegible]~~ that is ~~[illegible] stopp~~ kind of a while loop that iterates and take these coroutines.

→ It can only execute one ~~corride~~ coroutine get to a point when it stopped

→ While a coroutine is waiting (eg: I/O) the event loop can run other coroutines.

→ ~~So if there is a break and more is waiting~~

→ So if there is a break or IO is presenced the await keywords stop the function execution and starts the other function execution

→ ~~[illegible]~~ manual block as it is blocking something I/O gate.

→ super simple.

→ Planning agent, it ~~[illegible]~~ coming up with number of scenarios.

Coding → To create

OpenAI Agents SDK →

→ Introduction
① Lightweight and flexible
② Stay out of the way (opiniated)
③ make common activities easy.

OpenAI agents sdk terminologies →
① Agents represents LLMs
② Handoffs represent interactions b/w agents
③ Guardrails represents controls → It is used for the kinds of checks and control that you put around making.

→ In order to actually run an agent, we have three steps that i can uses →
① Create an Agents
② Use with trace() to the track agent.
③ call Runner.run() to run the agent

Vibe coding →

5 tips for creating good vibe coding
① Good vibes - prompt well · ask for short answers and latest APIs for todays date.
② Verify the vibe → ask of A+S question.
③ Step up the vibe → ask to break down your steps into independly testable steps
④ Vibe and validate → ask an LLM get another LLM to check
⑤ Vibe with variety → ask for 3 question to the same problem at,

→ Plan in an ag
coming up with number b

Coding → To create first agents

① from dotenv import load_dotenv
② from agents import agent, Runner, trace // important SDK fn

③ load_dotenv(override=True)
④ # Make an agent with name, instruction, model
⑤ agent = Agent(name="Jokester", instruction="You are a joke teller" model = "gpt-4-o-mini")

⑥ # instruction within agent is system prompt
⑦ agent
⑧ result = await Runner.run(agent, "Tell a joke about Autonomous AI agents")

⑨ print(result) # output coroutine object without await.
⑧ await ~~result~~
⑩ print(result.final_output) # gives a joke.

Q Does it means that OpenAI agents SDK always uses OpenAI's models?
A NO, definitely, the agents SDK can work with many other models.

⑪ # wrapping in trace (the result).
# above line 8.
⑫ with trace("Telling a joke"):
    result = await Runner.run (same as line 8)
    print(result.final_output)

tools
→ Plan an agent, it will take up the query remotely and
[illegible] up with number of searches.

Project 2 → Building a sales development rep.

We will build: (three different layers of Agentic Architecture):
- ① A workflow of Agent calls
- ② An agent that can use a tool.
- ③ An agent that can call on other agents
  Tools v/s Handoffs.

→ Sendgrid is used for setting up a company to send emails.
→ Sendgrid helps is to send transactional email from their servers.

[Note → @function_tool decorator converts a function into a function tool and it convert the function into json schema]

[Q What does that means to convert an agent into a tool?
Sol → It is going to create a new tool that is going to have all of the json schema that describes what that tool can do. and if we call the tool it's going to call the agent and make the agent make the call to llm's]

Difference/similarities b/w handoffs and Agents as tools :-
① similarties →
→ In both, an Agent can collaborate with another agent.
② Difference →
- In tools controls passes back to us and we can continue the execution as the main agent
→ In handoffs, after you are done the flows goes to the other agent's and it will come back to you.

→ Runner.run_streamed →
Runs the agent in streaming mode. Instead for waiting for the full AI response at once, it sends pieces of it as they're generated

→ async for event in result.stream_event() :

Day 3 → Multi-model Integration -

Objective →
① Models other than openAI → gemini, groq
② structured Output → the agent not to just respond with text, but with some desired fields
③ Guardrails → validation

→ while creating an agent in model field if you pass a string that it assumes that you working openAI model.

→ guardrails can only be applied either to the very begining of the input of the first agent or the output of the last agent.

Project 3 → Day 3 : Deep research project

→ Deep research is one of the classic uses cases of Agentic AI, the case where you have an agent that is able to go search the internet

→ This is well known since many of the frontier labs offer this agents via their online chat tools.

→ using Tools, structured Outputs, Hosted Tool.
tools running remotely

→ Planner agent, it will take up the query and coming up with number of searches.