Open AI Agents SDK: 40 Detailed Questions with Coding Examples (Roman Urdu)

Introduction

Ye document Open AI Agents SDK ke 40 important questions aur unke jawabaat cover karta hai, jo Roman Urdu mein hain aur beginners ke liye samajhna asaan hai. Har question ke saath detailed explanation aur coding example diya gaya hai, jo SDK ke concepts ko clear karta hai.

Question 1: Custom tool behavior functions ka main faida kya hai?

- **ANS**: Custom tool behavior functions agent ke tool usage ko control aur customize karte hain. Ye flexibility dete hain ke aap decide kar sakein ke kis tool ko kab call karna hai ya rokna hai, aur output ko kaise process karna hai. Ye complex workflows ke liye useful hai, jaise specific conditions check karna ya multiple tools ko coordinate karna.
- **Coding Example**:

```python
from agents import Agent, tool, Runner
@tool
def get_weather(city: str) -> str:
    return f"Weather in {city} is sunny!"
async def custom_behavior(ctx, agent, tool_output):
    if "sunny" in tool_output.output:
        return {"is_final_output": True, "output": "Great day for a
            walk!"}
    return {"is_final_output": False, "output": tool_output.output}
agent = Agent(name="WeatherBot", tools=[get_weather],
    tool_use_behavior=custom_behavior)
result = Runner.run_sync(agent, "Check weather in Karachi")
print(result.final_output)  # Output: Great day for a walk!
```

Question 2: Agent ko asynchronously execute karne ka konsa method hai?

- **ANS**: `Runner.run()` method agent ko asynchronously execute karta hai. Ye async/await syntax ke saath kaam karta hai, jo non-blocking execution ke liye ideal hai, khaas kar jab multiple tasks ya API calls involve hon.
- **Coding Example**:

```python
import asyncio
from agents import Agent, Runner
agent = Agent(name="AsyncBot", instructions="Answer quickly!")
async def main():
    result = await Runner.run(agent, "Say hello asynchronously")
    print(result.final_output)
asyncio.run(main())  # Output: hello
```

Question 3: RunContextWrapper ka maqsad kya hai?

- **ANS**: `RunContextWrapper` agent ke execution ke dauran context ko manage karta hai, jaise current state, tool outputs, aur handoff details. Ye ensure karta hai ke agent ko har step pe relevant info mile aur execution consistent rahe.
- **Coding Example**:

```python
from agents import RunContextWrapper, Agent
agent = Agent(name="ContextBot", instructions="Use context")
context = RunContextWrapper(agent=agent, input="User input")
print(context.get_input())  # Output: User input
```

Question 4: Runner.run_sync() kya karta hai?

- **ANS**: `Runner.run_sync()` agent ko synchronously chalata hai, yani task complete hone ka wait karta hai aur result return karta hai. Ye simple scripts ya testing ke liye useful hai jab async nahi chahiye.
- **Coding Example**:

```python
from agents import Agent, Runner
agent = Agent(name="SyncBot", instructions="Greet the user")
result = Runner.run_sync(agent, "Say hi")
print(result.final_output)  # Output: hi
```

Question 5: Pydantic model config mein extra=”forbid” kya karta hai?

- **ANS**: `extra="forbid"` Pydantic model mein ensure karta hai ke sirf defined fields hi accept hon. Agar koi extra field pass kiya jaye, toh error raise hota hai, jo data validation ko strict karta hai.
- **Coding Example**:

```python
from pydantic import BaseModel
class User(BaseModel, extra="forbid"):
    name: str
    age: int
try:
    user = User(name="Ali", age=25, extra_field="error")  # Raises
       ValidationError
except Exception as e:
    print(e)  # Output: extra fields not permitted
```

Question 6: Strict schemas ka non-strict ke muqable main faida kya hai?

- **ANS**: Strict schemas data ko tightly validate karte hain, errors ko kam karte hain, aur predictable outputs dete hain. Non-strict schemas extra fields allow karte hain, jo flexibility dete hain lekin errors ka risk barhate hain.
- **Coding Example**:

```python
from pydantic import BaseModel
class StrictUser(BaseModel, extra="forbid"):
    name: str
user = StrictUser(name="Ahmed")  # Works fine
# user = StrictUser(name="Ahmed", age=30)  # Error: extra fields not
    permitted
```

Question 7: StopAtTools ko multiple tool names ke saath combine karne se kya hota hai?

- **ANS**: `StopAtTools` specific tools ke set ke call hone par agent ko rukne deta hai. Ye selective execution ke liye useful hai jab aap chahate hain ke agent sirf designated tools ke baad hi pause kare.
- **Coding Example**:

```python
from agents import Agent, tool, Runner
@tool
def tool1(): return "Tool 1"
@tool
def tool2(): return "Tool 2"
agent = Agent(name="StopBot", tools=[tool1, tool2],
    tool_use_behavior={"type": "StopAtTools", "tool_names": ["tool1"
    ]})
result = Runner.run_sync(agent, "Run tool1")
print(result.final_output)  # Output: Tool 1
```

Question 8: OpenAI Agents SDK mein context ka maqsad kya hai?

- **ANS**: Context agent ke decisions aur tool calls ke liye background info deta hai, jaise user input, previous tool outputs, ya state. Ye agent ko relevant responses dene mein madad karta hai.
- **Coding Example**:

```python
from agents import Agent, RunContextWrapper
agent = Agent(name="ContextBot", instructions="Use context")
context = RunContextWrapper(agent=agent, input="Hello from Karachi")
print(context.get_input())  # Output: Hello from Karachi
```

Question 9: Tool_use_behavior modes ke darmiyan chunao ka key factor kya hai?

- **ANS**: Task ki complexity aur performance needs. `stop_on_first_tool` fast hai lekin limited, jabke `run_llm_again` flexible hai lekin slow, kyunke LLM ko multiple baar call karta hai.
- **Coding Example**:

```python
from agents import Agent, tool, Runner
@tool
def greet(): return "Hello!"
agent = Agent(name="BehaviorBot", tools=[greet], tool_use_behavior="
    stop_on_first_tool")
result = Runner.run_sync(agent, "Greet me")
print(result.final_output)  # Output: Hello!
```

Question 10: Handoff_description kya information deta hai?

- **ANS**: Ye batata hai ke task kyun aur kaise doosre agent ko delegate kiya gaya, jaise task ka purpose aur context, taake doosra agent usse samajh sake.
- **Coding Example**:

```python
from agents import Agent
agent1 = Agent(name="MainBot", instructions="Delegate tasks")
agent2 = Agent(name="HelperBot", handoff_description="Handles math
    tasks")
print(agent2.handoff_description)  # Output: Handles math tasks
```

Question 11: ToolsToFinalOutputResult.is_final_output kya indicate karta hai?

- **ANS**: Ye boolean batata hai ke tool ka output final hai ya agent ko aur processing karni hai. `True` ka matlab hai ke output final hai.
- **Coding Example**:

```python
from agents import ToolsToFinalOutputResult
result = ToolsToFinalOutputResult(output="Done", is_final_output=
    True)
print(result.is_final_output)  # Output: True
```

Question 12: Hosted tools aur function tools mein tool_use_behavior ka kya farq hai?

- **ANS**: Hosted tools externally managed hote hain (jaise OpenAI ke servers par), jabke function tools local Python functions hote hain, jo zyada customizable aur fast hote hain.
- **Coding Example**:

```python
from agents import tool
@tool
def local_tool(): return "Local result"
# Hosted tools ka example SDK ke hosted environment mein hota hai
```

Question 13: Agent ko doosre agents ke liye tool kaise banate hain?

- **ANS**: Agent ko `@tool` decorator se wrap kar ke uske functionality ko tool ke tor par define karte hain, jo doosre agents use kar sakte hain.
- **Coding Example**:

```python
from agents import Agent, tool
@tool
def agent_as_tool():
    agent = Agent(name="SubAgent", instructions="Return a number")
    return "42"
```

Question 14: Agent ke liye available sab tools kaunsa method return karta hai?

- **ANS**: `agent.get_tools()` method ek list return karta hai jisme agent ke available sab tools hote hain.
- **Coding Example**:

```python
from agents import Agent, tool
@tool
def sample_tool(): return "Tool"
agent = Agent(name="ToolBot", tools=[sample_tool])
print(agent.get_tools())  # Output: [<function sample_tool>]
```

Question 15: Har function tool ka pehla parameter kya hota hai?

- **ANS**: Har function tool ka pehla parameter `context` hota hai, jo execution state aur relevant info deta hai.
- **Coding Example**:

```python
from agents import tool
@tool
def my_tool(context, param: str):
    return f"Context: {context.get_input()}, Param: {param}"
```

Question 16: get_system_prompt() method ka maqsad kya hai?

- **ANS**: Ye agent ke default system prompt ko retrieve karta hai, jo uske behavior aur responses ko guide karta hai.
- **Coding Example**:

```python
from agents import Agent
agent = Agent(name="PromptBot", instructions="Be helpful")
print(agent.get_system_prompt())  # Output: Be helpful
```

Question 17: InputGuardrail aur OutputGuardrail mein kya farq hai?

- **ANS**: `InputGuardrail` user ke input ko validate karta hai (jaise malicious content check), jabke `OutputGuardrail` agent ke output ko validate karta hai (jaise correct format).

- **Coding Example**:

```python
from agents import Agent, GuardrailFunctionOutput
async def input_guardrail(ctx, agent, input):
    return GuardrailFunctionOutput(output_info=input,
        tripwire_triggered=False)
agent = Agent(name="GuardBot", input_guardrails=[input_guardrail])
```

Question 18: Agent ke instructions parameter ka primary maqsad kya hai?

- **ANS**: Instructions agent ka role aur behavior define karte hain, jaise ke wo ek customer support bot hai ya math solver.
- **Coding Example**:

```python
from agents import Agent, Runner
agent = Agent(name="Helper", instructions="Solve math problems")
result = Runner.run_sync(agent, "2 + 2")
print(result.final_output)  # Output: 4
```

Question 19: reset_tool_choice parameter kya control karta hai?

- **ANS**: Ye decide karta hai ke tool selection ko reset karna hai ya previous state se continue karna hai, khaas kar handoffs ke dauran.
- **Coding Example**:

```python
from agents import Agent
agent = Agent(name="ToolBot", reset_tool_choice=True)
# Tools reset honge har run ke liye
```

Question 20: Agar function tool exception raise karta hai, toh kya hota hai?

- **ANS**: Exception agent ke execution ko rok deta hai, aur error trace ke saath return hota hai. Aapko try-except use karna chahiye error handling ke liye.
- **Coding Example**:

```python
from agents import tool
@tool
def risky_tool():
    raise ValueError("Error occurred")
try:
    result = Runner.run_sync(Agent(tools=[risky_tool]), "Run risky
        tool")
except Exception as e:
    print(e)  # Output: Error occurred
```

Question 21: clone() method kya karta hai?

- **ANS**: `clone()` agent ka ek naya copy banata hai jo same configuration rakhta hai, lekin independent instance hota hai.
- **Coding Example**:

```python
from agents import Agent
agent = Agent(name="Original", instructions="Hello")
cloned = agent.clone()
print(cloned.name)  # Output: Original
```

Question 22: ToolsToFinalOutputFunction SDK mein kya hai?

- **ANS**: Ye ek function hai jo tool outputs ko final response mein convert karta hai, post-processing ke liye, jaise formatting ya filtering.

- **Coding Example**:

```
from agents import ToolsToFinalOutputFunction
def my_final_output(tool_outputs):
    return {"result": tool_outputs[0].output}
final_func = ToolsToFinalOutputFunction(my_final_output)
```

Question 23: Runner.run() ka return type kya hai?

- **ANS**: Ye `TaskResult` object return karta hai, jisme final output, tool results, aur execution details hote hain.
- **Coding Example**:

```
from agents import Agent, Runner
agent = Agent(name="TaskBot")
async def main():
    result = await Runner.run(agent, "Do something")
    print(type(result))  # Output: <class 'agents.TaskResult'>
asyncio.run(main())
```

Question 24: Agar custom tool behavior function is_final_output=False return karta hai, toh kya hota hai?

- **ANS**: Agent processing continue karta hai, LLM ko dobara call karta hai, aur further tools ya actions execute karta hai.
- **Coding Example**:

```
from agents import Agent, tool
@tool
def step1(): return "Step 1 done"
async def custom_behavior(ctx, agent, tool_output):
    return {"is_final_output": False, "output": tool_output.output}
agent = Agent(tools=[step1], tool_use_behavior=custom_behavior)
```

Question 25: StopAtTools ka stop_on_first_tool ke bajaye kab use karte hain?

- **ANS**: `StopAtTools` jab use hota hai jab aap specific tools ke set par rukna chahte hain, na ke pehle tool par, taake selective control mile.
- **Coding Example**:

```
from agents import Agent, tool
@tool
def tool1(): return "Tool 1"
agent = Agent(tools=[tool1], tool_use_behavior={"type": "StopAtTools
    ", "tool_names": ["tool1"]})
```

Question 26: Agent mein tool_use_behavior ka default value kya hai?

- **ANS**: Default value `run_llm_again` hai, jo LLM ko multiple tool calls ke liye dobara chalata hai taake complex tasks handle hon.
- **Coding Example**:

```
from agents import Agent
agent = Agent(name="DefaultBot")  # tool_use_behavior is
    run_llm_again
```

Question 27: run_llm_again aur stop_on_first_tool mein performance ka key farq kya hai?

- **ANS**: `run_llm_again` flexible hai lekin slow kyunke LLM ko baar baar call karta hai. `stop_on_first_tool` fast hai kyunke pehle tool par ruk jata hai.

- **Coding Example**:

```python
from agents import Agent, tool
@tool
def fast_tool(): return "Done"
agent = Agent(tools=[fast_tool], tool_use_behavior="
    stop_on_first_tool")
```

Question 28: Pydantic v2 mein field validation ke liye kaunsa decorator use hota hai?

- **ANS**: `@field_validator` decorator field-level validation ke liye use hota hai, jo custom logic se data check karta hai.
- **Coding Example**:

```python
from pydantic import BaseModel, field_validator
class User(BaseModel):
    age: int
    @field_validator("age")
    def check_age(cls, v):
        if v < 18: raise ValueError("Age must be 18+")
        return v
```

Question 29: Agent mein model_settings ka maqsad kya hai?

- **ANS**: `model_settings` LLM ke configurations, jaise model type, temperature, ya max tokens, ko customize karta hai taake agent ka behavior adjust ho.
- **Coding Example**:

```python
from agents import Agent
agent = Agent(name="CustomBot", model_settings={"model": "gpt-4", "
    temperature": 0.7})
```

Question 30: Flexible schemas ke liye non-strict mode kaise enable karte hain?

- **ANS**: Pydantic model mein `extra="allow"` set karo taake extra fields accept hon, jo flexible data handling ke liye useful hai.
- **Coding Example**:

```python
from pydantic import BaseModel
class FlexibleUser(BaseModel, extra="allow"):
    name: str
user = FlexibleUser(name="Ali", extra_field="value")  # No error
```

Question 31: handoff_description parameter kya karta hai?

- **ANS**: Ye task delegation ke purpose aur context ko define karta hai, taake doosra agent task ko samajh sake aur uspe action le.
- **Coding Example**:

```python
from agents import Agent
agent = Agent(name="MathBot", handoff_description="Handles complex
    calculations")
print(agent.handoff_description)  # Output: Handles complex
    calculations
```

Question 32: Schema evolution ko backward compatibility ke saath kaise implement karte hain?

- **ANS**: Naye fields ko optional banayein ya default values dein, aur `extra="allow"` use karo taake purane schemas ke saath compatibility rahe.

- **Coding Example**:

```python
from pydantic import BaseModel
class UserV2(BaseModel, extra="allow"):
    name: str
    age: int = None  # Optional for backward compatibility
```

Question 33: Kya dynamic instructions async functions ho sakte hain?

- **ANS**: Haan, dynamic instructions async functions ho sakte hain, jo context-based updates ke liye useful hain, khaas kar real-time applications mein.
- **Coding Example**:

```python
from agents import Agent
async def dynamic_instructions(context):
    return "Dynamic instruction based on " + context.get_input()
agent = Agent(name="DynamicBot", instructions=dynamic_instructions)
```

Question 34: Jab tool_use_behavior 'stop_on_first_tool' set hota hai, toh kya hota hai?

- **ANS**: Agent pehle tool call ke baad ruk jata hai aur output return karta hai, bina LLM ko dobara chalaye, jo fast execution ke liye hai.
- **Coding Example**:

```python
from agents import Agent, tool
@tool
def quick_tool(): return "Quick result"
agent = Agent(tools=[quick_tool], tool_use_behavior="
    stop_on_first_tool")
```

Question 35: Mutable aur immutable context patterns mein kya farq hai?

- **ANS**: Mutable context execution ke dauran modify ho sakta hai, jabke immutable fixed rahta hai aur change nahi hota, jo consistency ke liye hai.
- **Coding Example**:

```python
from agents import RunContextWrapper
context = RunContextWrapper(agent=Agent(name="Bot"), input="Test")
context.set("key", "value")  # Mutable context example
```

Question 36: 'additionalProperties should not be set for object types' error ka kya wajah hai?

- **ANS**: Ye error tab hota hai jab strict schema (`extra="forbid"`) mein extra fields pass kiye jate hain, jo model ke rules ke khilaf hai.
- **Coding Example**:

```python
from pydantic import BaseModel
class StrictModel(BaseModel, extra="forbid"):
    name: str
# StrictModel(name="Ali", extra="value")  # Raises error
```

Question 37: Non-strict schemas ko strict schemas ke bajaye kab use karte hain?

- **ANS**: Non-strict schemas jab use hote hain jab input data unpredictable ho ya future mein new fields add hone ki possibility ho, taake flexibility mile.
- **Coding Example**:

```python
from pydantic import BaseModel
class NonStrictModel(BaseModel, extra="allow"):
    name: str
model = NonStrictModel(name="Ali", extra="value")  # Works fine
```

Question 38: Custom tool behavior function ko kaunse parameters milte hain?

- **ANS**: Isse `context`, `agent`, aur `tool_output` parameters milte hain, jo execution state, agent details, aur tool ka result dete hain.
- **Coding Example**:

```python
from agents import Agent, tool
@tool
def sample_tool(): return "Tool result"
async def custom_behavior(context, agent, tool_output):
    return {"is_final_output": True, "output": f"Processed: {
        tool_output.output}"}
```

Question 39: Runner.run_streamed() kya return karta hai?

- **ANS**: Ye ek generator object return karta hai jo real-time mein partial results stream karta hai, useful for long-running tasks.
- **Coding Example**:

```python
from agents import Agent, Runner
agent = Agent(name="StreamBot")
async def main():
    async for result in Runner.run_streamed(agent, "Stream this"):
        print(result)
asyncio.run(main())
```

Question 40: Context ke mutabiq dynamic instructions kaise banate hain?

- **ANS**: `get_system_prompt()` ko override karo ya async function use karo jo context ke hisaab se instructions dynamically update kare.
- **Coding Example**:

```python
from agents import Agent, Runner
async def dynamic_prompt(context):
    return f"Respond based on: {context.get_input()}"
agent = Agent(name="DynamicBot", instructions=dynamic_prompt)
result = Runner.run_sync(agent, "User input")
print(result.final_output)  # Output based on input
```