OpenAI Agents SDK Basic to Intermediate Questions

Q1. What's the main advantage of custom tool behavior functions?

Answer: This setting controls whether the tool result is sent to the LLM or used directly as the final output. For FunctionTools, you can choose default, stop after the first tool, or use a custom function. Hosted tools are always processed by the LLM.

Q2. Which method is used to execute an agent asynchronously?

Answer: `Runner.run()` is use to run an agent asynchronously.

Q3. What's the purpose of RunContextWrapper?

Answer: It gives access to the context and important run details during the agent's work. It wraps the object you pass to `Runner.run()`, and helps tools or agents use that data.

Q4. What does `Runner.run_sync()` do?

Answer: `Runner.run_sync()` is a synchronous method that runs `run()` internally.

Q5. What does extra="forbid" do in a Pydantic model config?

Answer: `extra="forbid"` disallows any extra fields not defined in the Pydantic model. If you try to pass extra data, it will raise an error.

Q6. What's the main advantage of strict schemas over non-strict?

Answer: Strict schemas help ensure that the tool receives correctly formatted and valid JSON input. This reduces errors and makes the tool more reliable.

Q7. What happens when you combine `StopAtTools` with multiple tool names?

Answer: When you provide multiple tool names to `StopAtTools`, the agent stops as soon as any of those tools is called and returns that tool's result as the final output.

Q8. What is the purpose of context in the OpenAI Agents SDK?

Answer: The purpose of context in the OpenAI Agents SDK is to provide shared data and tools (like user info, helper functions, or settings) that the agent, tools, and hooks can use during a run. This context is not visible to the LLM. If you want the LLM to see some data, you must include it in the system prompt, input, or tool output.

Q9. What's the key consideration when choosing between different tool_use_behavior modes?

Answer: The key consideration is whether you want the tool result to be sent back to the LLM for further reasoning, or used directly as the final output.This choice affects how the agent responds — whether it stops after the tool or continues reasoning using the LLM.

Q10. What information does handoff_description provide?

Answer: `handoff_description` is a short description that tells the LLM what the agent does and when to use it. It helps the LLM know that this agent is for handoff.

Q11. What does ToolsToFinalOutputResult.is_final_output indicate?

Answer: It checks the result is final or not. If its not, it sent back to LLM to continue the conversation.

Q12. What's the difference between hosted tools and function tools in terms of tool_use_behavior?

Answer: Function tools are Python functions whose behavior we can control. Hosted tools are OpenAI-provided tools that are always controlled by the LLM, and we cannot customize their behavior.

Q13. How do you convert an agent into a tool for other agents?

Answer: `agent.as_tool()` is a function that converts an agent into a tool so other agents can use it.

Q14. What method returns all tools available to an agent?

Answer: `agent.tools` returns all tools available to an agent.

Q15. What is the first parameter of every function tool?

Answer: The first parameter of every custom function tool is the `context`.

Q16. What is the purpose of the `get_system_prompt()` method?

Answer: It returns the system prompt (instructions) for the agent.

Q17. What's the difference between InputGuardrail and OutputGuardrail?

Answer: Both guardrails follow the same steps, but the Input Guardrail checks the user's input, while the Output Guardrail checks the agent's output.

Q18. What is the primary purpose of the `instructions` parameter in an Agent?

Answer: The `instructions` parameter acts as the system prompt. It tells the agent what to do and how to respond when it is called.

Q19. What does the `reset_tool_choice` parameter control?

Answer: It resets the tool choice to "auto" after a tool is called, to prevent the agent from going into an infinite loop.

Q20. What happens if a function tool raises an exception?

Answer: If a function tool raises an error, the agent will use a default error function to tell the LLM. You can also give your own error function to send a custom message instead. If you pass `None`, the error will not be handled and must be caught by your own code.

Q21. What does the `clone()` method do?

Answer: The `clone()` method duplicates an agent and allows you to modify its properties.

Q22. What is ToolsToFinalOutputFunction in the OpenAI Agents SDK?

Answer: It is a function that checks result and return final output.

Q23. What is the return type of `Runner.run()`?

Answer: The return type of `Runner.run()` is `RunResult`, which holds the agent's final output.

Q24. What happens if a custom tool behavior function returns `is_final_output=False`?

Answer: If a custom tool behavior function returns `is_final_output=False`, the agent will not treat the output as final and will continue the conversation by passing the result back to the LLM for further reasoning.

Q25. When would you use StopAtTools instead of stop_on_first_tool?

Answer: We use `StopAtTools` when we want to stop the agent after calling specific tools. This is better than using `stop_on_first_tool`, which stops the agent after the first tool call, no matter which tool it is.

Q26. What is the default value for tool_use_behavior in an Agent?

Answer: The default value of `tool_use_behavior` is "`run_llm_again`".

Q27. What's the key difference between `run_llm_again` and `stop_on_first_tool` in terms of performance?

Answer: `run_llm_again` sends the tool result back to the LLM, while `stop_on_first_tool` directly uses the first tool result as the final output.

Q28. Which Pydantic v2 decorator is used for field validation?

Answer: In Pydantic v2, the decorator used for field validation is `@field_validator`.

Q29. What is the purpose of `model_settings` in an Agent?

Answer: `model_settings` is used to configure the model globally, such as temperature, top_p, etc. These settings override the default model behavior for the agent.

Q30. How do you enable non-strict mode for flexible schemas?

Answer: Use extra="allow" in Pydantic model config.

Q31. What does the handoff_description parameter do?

Answer: Tells the LLM that the agent is a handoff and helps it decide when to use it.

Q32. How do you implement schema evolution while maintaining backward compatibility?

Answer: Use optional fields and default values in your models to support older data formats.

Q33. Can dynamic instructions be async functions?

Answer: Yes, dynamic instructions support both regular and async functions.

Q34. What happens when `tool_use_behavior` is set to `'stop_on_first_tool'`?

Answer: When `tool_use_`behavior is set to `stop_on_first_tool`, the agent immediately stops after calling the first tool and returns that tool's result as the final output.

Q35. What’s the difference between mutable and immutable context patterns?

Answer: Mutable context can be changed during the run.
Immutable context stays the same — you only read from it.

Q36. What causes the error 'additionalProperties should not be set for object types'?

Answer: This error happens if you include `additionalProperties` in a `params_json_schema`. The SDK doesn’t allow setting it manually for object types.

Q37. When should you use non-strict schemas over strict schemas?

Answer: When you need more flexibility in the input and can handle incorrect data easily.

Q38. In a custom tool behavior function, what parameters does it receive?

Answer: It receives the context and the arguments (as JSON string).

Q39. What does `Runner.run_streamed()` return?

Answer: `Runner.run_streamed()` returns a `RunResultStreaming` object.

Q40. How do you create dynamic instructions that change based on context?

Answer: Use a function (sync or async) that returns instructions using the context.

Prepared by UMM-E-HANI