AGENTS

Q1. What are *agents* in the context of AI apps?
A) A database to store data
B) The core building block in apps, based on LLMs
C) A visualization tool for AI models
D) A programming framework for APIs
Answer: B

Q2. Which of the following is **required** when configuring an agent?
A) Instructions
B) Name
C) Tools
D) Model settings
Answer: B

Q3. What is another name for **instructions** in an agent?
A) Developer message or system prompt
B) User query
C) Model configuration
D) Tool function
Answer: A

Q4. Which parameter controls model tuning like *temperature* and *top_p*?
A) instructions
B) model_settings
C) name
D) tools
Answer: B

Q5. In the given code example, which tool is implemented?
A) Translation tool
B) Math calculator tool
C) Weather information tool
D) Haiku generator tool
Answer: C

Q6. In the code, what is the **role of @function_tool decorator**?
A) To define the model type
B) To specify the LLM instructions
C) To convert a function into a tool usable by the agent
D) To generate random outputs
Answer: C

Q7. What is the instruction given to the "Haiku agent"?
A) Always return weather info
B) Always respond in haiku form
C) Always generate JSON output
D) Always provide short answers
Answer: B

Q8. Which model is used in the example?
A) gpt-4
B) gpt-3.5
C) o3-mini
D) o1-preview
Answer: C
Q1. An **AGENT** in AI applications is primarily:
A) A database schema
B) A large language model configured with instructions and tools
C) A middleware library for APIs
D) A user input parser
Answer: B

Q2. Which of the following are **common properties** of an agent?
A) Name
B) Instructions (system prompt)
C) Model & model_settings
D) Tools
E) All of the above
Answer: E

Q3. True or False:
The *instructions* of an agent are also referred to as a **developer message** or **system prompt**.
Answer: True

Q4. In model configuration, which setting(s) can influence randomness and diversity of responses?
A) Temperature
B) top_p
C) Both A and B
D) None of the above
Answer: C

Q5. In the provided code, what does the get_weather function do?
A) Returns temperature of a city
B) Returns a hardcoded response "sunny" for the specified city
C) Calls an external API for weather updates
D) Generates haiku poems
Answer: B

Q6. What is the purpose of the @function_tool decorator?
A) It marks a function so the agent can use it as a tool
B) It defines the model to be used
C) It automatically optimizes LLM responses
D) It sets default temperature and top_p values
Answer: A

Q7. In the example, the **agent name** is:
A) Weather Agent
B) Haiku Agent
C) o3-mini Agent
D) Sunny Agent
Answer: B

Q8. The instruction "Always respond in haiku form" tells the agent to:
A) Always return weather details
B) Always generate code snippets
C) Always answer in a haiku poetry style
D) Always output JSON
Answer: C

Q9. Which model is used in the given code snippet?
A) gpt-4
B) gpt-3.5
C) o3-mini
D) o1-preview
Answer: C

Q10. True or False:
The tools parameter in the Agent configuration is optional and can be left empty if not required.
Answer: True

Q1. Which of the following is **NOT** a core property of an Agent?
A) Name
B) Instructions
C) Model
D) Database schema
Answer: D

Q2. The role of **@function_tool** in the code is to:
A) Automatically train the model
B) Convert a Python function into a tool usable by the agent
C) Set the model temperature
D) Define the agent's name
Answer: B

Q3. The instruction "Always respond in haiku form" makes the agent:
A) Return only weather data
B) Output answers in haiku poetry style
C) Generate JSON results
D) Ignore tools
Answer: B

Q4. Which settings can affect **randomness and creativity** of an agent's output?
A) Temperature
B) top_p
C) Both A and B
D) None of the above
Answer: C

Q5. True or False:
The tools property in Agent configuration is always mandatory.
Answer: False

Context

Q1. In Agents, what is the role of **Context**?
A) It defines the agent's name and model
B) It acts as a dependency-injection tool, carrying state and dependencies
C) It stores only user input prompts
D) It controls temperature and top_p values
Answer: B

Q2. How is **Context** provided to agents during execution?
A) Passed directly inside the agent constructor
B) Passed to Runner.run() and then injected into every agent, tool, and handoff
C) Stored inside model_settings
D) Attached as system prompt
Answer: B

Q3. Which of the following can be used as a Context object?
A) Only predefined classes
B) Only dataclasses
C) Any Python object
D) Only dictionaries
Answer: C

Q4. In the given UserContext example, which attributes are included?
A) name, uid, is_pro_user
B) temperature, top_p
C) city, weather
D) calendar, participants
Answer: A

Q5. By default, what type of output do agents produce?
A) JSON
B) str (plain text)
C) Pydantic object
D) TypedDict
Answer: B

Q6. Which parameter allows customizing the output type of an agent?
A) model_settings
B) output_type
C) instructions
D) tools
Answer: B

Q7. What kinds of types can be wrapped for use as output_type?
A) Only Pydantic models
B) Only dataclasses
C) Only TypedDict
D) Any type supported by Pydantic TypeAdapter (dataclasses, lists, TypedDict, etc.)
Answer: D

Q8. In the given code, what is the purpose of the **CalendarEvent** class?
A) To store user purchases
B) To represent extracted calendar events using a structured schema
C) To define agent instructions
D) To configure model settings
Answer: B

Q9. In the example, the agent named **"Calendar extractor"** is instructed to:
A) Fetch purchases
B) Extract calendar events from text
C) Return weather updates
D) Identify pro users
Answer: B

Q10. True or False:
Agents can only produce text outputs, not structured objects.
Answer: False

Q1. The **Context** in agents can be best described as:
A) A storage of user prompts only
B) A dependency-injection tool carrying state and dependencies
C) A JSON object returned by the agent
D) A logging framework
Answer: B

Q2. True or False:
You can provide **any Python object** as the context.
Answer: True

Q3. By default, agents produce outputs of type:
A) JSON
B) str (plain text)
C) Pydantic object
D) TypedDict
Answer: B

Q4. The parameter used to specify a structured output type for agents is:
A) model_settings
B) tools
C) output_type
D) instructions
Answer: C

Q5. In the "Calendar extractor" example, the agent is configured to:
A) Fetch purchase history
B) Extract calendar events from text into a structured object
C) Generate haiku poems
D) Manage user context for authentication
Answer: B

dynamic instructions

Q1. What is the main purpose of **dynamic instructions** in an agent?
A) To permanently set fixed rules for the agent
B) To allow generating instructions at runtime using a function
C) To increase the randomness of model outputs
D) To automatically tune model parameters
Answer: B

Q2. When using dynamic instructions, the function must:
A) Return a JSON schema
B) Return the prompt (string) for the agent
C) Return the context object directly
D) Only return an integer
Answer: B

Q3. What arguments does a dynamic instruction function receive?
A) agent and tools
B) model and context
C) agent and context
D) name and output_type
Answer: C

Q4. Which of the following is true about dynamic instruction functions?
A) Only regular functions are supported
B) Only async functions are supported
C) Both regular and async functions are accepted
D) Neither are supported
Answer: C

Q5. In the example, what does the dynamic_instructions function return?
A) A user's purchase list
B) The agent's model name
C) A string prompt including the user's name
D) A hardcoded weather report
Answer: C

Q6. In the example, what type of agent is created?
A) Calendar extractor agent
B) Haiku agent
C) Triage agent
D) Refund agent
Answer: C

Q7. True or False:
Dynamic instructions completely replace static instructions that can be provided when creating an agent.
Answer: False (you can use either static or dynamic depending on the need).

Q1. The key advantage of **dynamic instructions** is:
A) They allow runtime generation of instructions based on context
B) They make agents faster
C) They remove the need for a model parameter
D) They replace tools automatically
Answer: A

Q2. True or False:
Dynamic instruction functions must always be **async**.
Answer: False (they can be regular or async).

Q3. A dynamic instruction function must return:
A) A Python object
B) A JSON schema
C) A string prompt
D) A TypedDict
Answer: C

Q4. In the example, the dynamic instructions included:
A) The user's name from context
B) The agent's tool list
C) The refund agent's output
D) The model temperature
Answer: A

Q5. In the code, which agent uses dynamic instructions?
A) Calendar extractor
B) Haiku agent
C) Triage agent
D) Refund agent
Answer: C

MCQs on Running Agents
Q1. Which class is used to run agents?
A) AgentRunner
B) Runner
C) Executor
D) AgentManager
Answer: B

Q2. What does Runner.run() do?
A) Runs sync and returns plain text
B) Runs async and returns a **RunResult**
C) Runs only system prompts
D) Runs with no return value
Answer: B

Q3. What is the difference between Runner.run() and Runner.run_sync()?
A) run_sync() runs async, while run() is sync
B) run_sync() is a sync wrapper around .run()
C) run() streams outputs, while run_sync() does not
D) They are completely different methods with unrelated outputs
Answer: B

Q4. What does Runner.run_streamed() return?
A) Nothing
B) A string only
C) RunResultStreaming
D) JSON output only
Answer: C

Q5. What is the unique feature of Runner.run_streamed() compared to the others?
A) It always runs synchronously
B) It streams events as they are received from the LLM
C) It cannot handle text prompts
D) It does not support async execution
Answer: B

Q6. In the example, what type of agent is created?
A) Triage agent
B) Calendar extractor
C) Assistant agent
D) Refund agent
Answer: C

Q7. In the example, what kind of poem did the agent generate?
A) Sonnet
B) Free verse
C) Haiku
D) Epic
Answer: C

Q8. True or False:
Runner.run() and Runner.run_streamed() are both asynchronous.
Answer: True

Q1. The method Runner.run() is:
A) Sync and returns plain text
B) Async and returns a **RunResult**
C) Only for streaming
D) Deprecated
Answer: B

Q2. True or False:
Runner.run_sync() is just a synchronous wrapper around Runner.run().
Answer: True

Q3. Which method is used when you want **live streaming of outputs** from the LLM?
A) Runner.run()
B) Runner.run_sync()
C) Runner.run_streamed()
D) Runner.stream()
Answer: C

Q4. What does Runner.run_streamed() return?
A) RunResultStreaming
B) JSON only
C) A plain string
D) Nothing
Answer: A

Q5. In the provided code, the agent's final output was:
A) A refund request
B) A haiku about recursion in programming
C) A weather report
D) A calendar event extraction
Answer: B

MCQs on The Agent Loop

Q1. When using the Runner.run() method, what can the **input** be?
A) Only a string
B) Only a dictionary
C) A string (user message) or a list of input items (OpenAI Responses API items)
D) Only a JSON object
Answer: C

Q2. In the agent loop, what is the **first step**?
A) Check for max turns
B) Call the LLM for the current agent with the current input
C) Run handoffs
D) Execute tool calls
Answer: B

Q3. What happens if the LLM returns a **final_output**?
A) The loop continues with a new agent
B) The loop ends and returns the result
C) The loop raises an exception
D) The loop switches to tool execution
Answer: B

Q4. If the LLM does a **handoff**, what happens next?
A) The loop ends immediately
B) The current agent and input are updated, and the loop re-runs
C) A MaxTurnsExceeded exception is raised
D) The output is ignored
Answer: B

Q5. If the LLM produces **tool calls**, how does the loop handle them?
A) Ignores them and continues
B) Runs the tool calls, appends results, and re-runs the loop
C) Raises an exception
D) Switches to handoff mode
Answer: B

Q6. What happens if the loop exceeds the **max_turns** parameter?
A) It restarts with a new agent
B) It raises a **MaxTurnsExceeded** exception
C) It returns an empty result
D) It ends silently with no output
Answer: B

Q7. True or False:
The agent loop can handle **final outputs, handoffs, and tool calls** dynamically until completion or max_turns is reached.
Answer: True

Q1. The input to Runner.run() can be:
A) Only a string
B) A string or a list of OpenAI Responses API items
C) Only JSON objects
D) Only tool calls
Answer: B

Q2. True or False:
The very first step in the agent loop is to **call the LLM** for the current agent with the given input.
Answer: True

Q3. If the LLM produces a **final_output**, the loop:
A) Ends and returns the result
B) Switches to handoff mode
C) Ignores it and continues
D) Raises an exception
Answer: A

Q4. What happens when the LLM produces **tool calls**?
A) They are skipped, and the loop continues
B) They are executed, results appended, and the loop re-runs
C) They cause the loop to end
D) They reset the agent
Answer: B

Q5. What exception is raised if the loop exceeds the allowed number of turns?
A) MaxRetriesExceeded
B) MaxTurnsExceeded
C) TooManyStepsError
D) AgentLoopBreak
Answer: B

MCQs on Streaming

Q1. What does **streaming** allow in the context of agents?
A) To run multiple agents in parallel
B) To receive streaming events as the LLM runs
C) To convert outputs into JSON automatically
D) To bypass the agent loop
Answer: B

Q2. Once the stream is completed, which object contains the full information about the run?
A) RunResult
B) RunResultStreaming
C) StreamResult
D) AgentLoopResult
Answer: B

Q3. Which method is used to access streaming events?
A) .stream_output()
B) .get_events()
C) .stream_events()
D) .fetch_stream()
Answer: C

Q4. True or False:
RunResultStreaming contains only partial outputs and not the complete run information.
Answer: False (it contains the **complete information** after the stream finishes).

Q5. The main advantage of streaming is:
A) Lower token costs
B) Faster LLM training
C) Real-time access to outputs as they are generated
D) Avoiding handoffs
Answer: C

Q1. The main purpose of **streaming** is to:
A) Reduce token usage
B) Receive outputs in real-time as the LLM runs
C) Run multiple agents in sequence
D) Bypass tool execution
Answer: B

Q2. True or False:
When the stream is finished, **RunResultStreaming** contains the full details of the run.
Answer: True

Q3. Which method allows you to iterate through streaming events?
A) .get_events()
B) .stream_events()
C) .events()
D) .run_stream()
Answer: B

Q4. Compared to a normal run, the benefit of streaming is:
A) Outputs arrive incrementally before the run completes
B) Outputs are always shorter
C) Outputs skip the agent loop
D) Outputs can only be JSON
Answer: A

Q5. If you want to see partial responses while the LLM is still generating, which approach should you use?
A) Runner.run_sync()
B) Runner.run()
C) Streaming with RunResultStreaming
D) Static instructions only
Answer: C

MCQs on Run Config
Q1. What is the purpose of the **run_config** parameter?
A) To store only agent names
B) To configure global settings for an agent run
C) To disable all tool calls permanently
D) To create user context dynamically
Answer: B

Q2. Which option in run_config allows overriding the LLM model used by agents?
A) model_provider
B) model
C) model_settings
D) workflow_name
Answer: B

Q3. What does **model_provider** specify?
A) The database for storing results
B) The API key to use
C) A provider for looking up model names (defaults to OpenAI)
D) The agent handoff filter
Answer: C

Q4. Which configuration can override agent-specific parameters such as temperature or top_p?
A) model
B) workflow_name
C) model_settings
D) trace_metadata
Answer: C

Q5. What do **input_guardrails** and **output_guardrails** do?
A) Apply global validation or filtering to all inputs and outputs
B) Control token limits
C) Automatically disable tracing
D) Replace model settings
Answer: A

Q6. What is the role of **handoff_input_filter** in run_config?
A) Filters final outputs
B) Edits inputs sent to new agents during handoffs
C) Prevents tools from executing
D) Forces the agent to restart
Answer: B

Q7. Which flag allows disabling tracing for the entire run?
A) trace_id
B) tracing_disabled
C) trace_metadata
D) workflow_name
Answer: B

Q8. What does **trace_include_sensitive_data** control?
A) Whether to skip all traces
B) Whether traces include sensitive information like inputs/outputs
C) Whether only outputs are stored
D) Whether to enable guardrails globally
Answer: B

Q9. Which parameter should you **at least set** when configuring tracing?
A) trace_metadata
B) group_id
C) workflow_name
D) model_provider
Answer: C

Q10. What does **group_id** do in run_config tracing?
A) Links traces across multiple runs
B) Stores metadata for one run
C) Sets the model provider globally
D) Disables tracing
Answer: A

Q11. True or False:
trace_metadata allows attaching additional metadata to all traces.
Answer: True

Q1.
If you want to apply the same **temperature** to every agent in a run, which run_config setting should you use?
A) model
B) model_settings
C) model_provider
D) workflow_name
Answer: B

Q2.
Which run_config option lets you **edit inputs before passing them to a new agent during handoff**?
A) input_guardrails
B) handoff_input_filter
C) trace_metadata
D) model_provider
Answer: B

Q3.
To completely turn off tracing in a run, you would set:
A) workflow_name = None
B) tracing_disabled = True
C) trace_id = ""
D) model_provider = ""
Answer: B

Q4.
Which of the following is **recommended** to always set when configuring tracing?
A) trace_metadata
B) workflow_name
C) group_id
D) model
Answer: B

Q5. (Trick Question 🤔)
True or False:
Setting trace_include_sensitive_data = False will prevent traces from being generated.
Answer: False ✅ (It only controls whether sensitive info is included, not whether tracing happens at all.)

handoffs?
Q1. In Agents, what are **handoffs**?
A) A logging system for agent responses
B) Sub-agents that the main agent can delegate tasks to
C) Tools used to fetch external data
D) Instructions provided to the model
Answer: B

Q2. Why are handoffs considered a **powerful pattern**?
A) They simplify environment variable management
B) They allow orchestration of modular, specialized agents
C) They reduce API costs automatically
D) They increase token limits
Answer: B

Q3. How are handoffs provided to an agent?
A) As part of instructions
B) As part of model_settings
C) In a list assigned to the handoffs parameter
D) In the system prompt only
Answer: C

Q4. In the example, what is the role of the **triage_agent**?
A) To manage user authentication
B) To decide whether to handle the question itself or delegate to booking/refund agents
C) To process payments directly
D) To store conversation history
Answer: B

Q5. If a user asks about booking, what will the **triage_agent** do?
A) Answer itself using a tool
B) Ignore the query
C) Delegate the query to the booking_agent
D) Send the query to refund_agent
Answer: C

Q6. True or False:
Handoffs allow creating **specialized agents** that each excel at a single task.
Answer: True

Q7. Which agents are included in the handoffs list of the example?
A) triage_agent only
B) booking_agent and refund_agent
C) Calendar extractor and Haiku agent
D) None
Answer: B

Q1. The main purpose of **handoffs** is to:
A) Log user queries
B) Delegate tasks to specialized sub-agents
C) Increase the model temperature
D) Provide environment variables
Answer: B

Q2. True or False:
Handoffs allow the creation of modular agents, each focusing on a **single specialized task**.
Answer: True

Q3. In the example, the **triage_agent**:
A) Always answers queries itself
B) Routes user questions to booking_agent or refund_agent when relevant
C) Only handles refund queries
D) Stores purchase history
Answer: B

Q4. How are handoffs added to an agent?
A) By passing them inside instructions
B) By listing them in the handoffs parameter
C) By registering them as tools
D) By setting output_type
Answer: B

Q5. If a user asks about **refunds**, which agent will the triage_agent delegate to?
A) booking_agent
B) refund_agent
C) triage_agent itself
D) Calendar extractor agent
Answer: B

MCQs on Handoffs in Agents SDK
Q1. What is the main purpose of **handoffs** in an agent system?
a) To increase the speed of processing
b) To allow an agent to delegate tasks to another agent
c) To reduce the size of the LLM model
d) To encrypt communication between agents
Answer: b) To allow an agent to delegate tasks to another agent

---

Q2. Which of the following is an example scenario where **handoffs** are useful?
a) Translating documents
b) Playing background music
c) Customer support app with agents for order status, refunds, and FAQs
d) Image classification task
Answer: c) Customer support app with agents for order status, refunds, and FAQs

---

Q3. In the Agents SDK, how are **handoffs** represented to the LLM?
a) As functions
b) As tools
c) As datasets
d) As prompts
Answer: b) As tools

Q4. If there is a handoff to an agent named **Refund Agent**, what would the corresponding tool be called?
a) refund_tool_agent
b) handoff_refund
c) transfer_to_refund_agent
d) refund_transfer_tool
Answer: c) transfer_to_refund_agent

Q5. Which parameter do all agents have to support handoffs?
a) tasks
b) handoffs
c) delegate
d) agent_tools
Answer: b) handoffs

Q6. What does the **handoffs param** in an agent accept?
a) Only strings
b) An Agent directly or a Handoff object
c) Only functions
d) Only input filters
Answer: b) An Agent directly or a Handoff object

Q7. Which function in the Agents SDK is used to create a handoff?
a) transfer()
b) handoff()
c) delegate()
d) switch_agent()
Answer: b) handoff()

Q8. What can you specify when creating a handoff using the **handoff() function**?
a) Just the agent name
b) Agent to hand off to, optional overrides, and input filters
c) Only overrides
d) Only input filters
Answer: b) Agent to hand off to, optional overrides, and input filters

---

Q9. In the example, which agents are created besides the **triage agent**?
a) Refund agent and Order agent
b) Billing agent and Refund agent
c) FAQ agent and Refund agent
d) Customer agent and Billing agent
Answer: b) Billing agent and Refund agent

Q10. The **triage agent** in the example uses handoffs to:
a) Refund agent only
b) Billing agent only
c) Billing agent and Refund agent
d) No other agent
Answer: c) Billing agent and Refund agent

MCQs
Q1. What is the purpose of using *handoff inputs* in an agent setup?
A) To stop an agent from running
B) To provide additional structured data when calling another agent
C) To delete logs after execution
D) To prevent API key usage
Answer: B

Q2. In the given example, what type of data is expected to be passed during the handoff to the *Escalation agent*?
A) Integer values
B) Boolean values
C) String reason wrapped in a model
D) JSON without validation
Answer: C

Q3. Which library is used to define the EscalationData model?
A) dataclasses
B) pydantic
C) marshmallow
D) fastapi
Answer: B

Q4. What does the function on_handoff do in this example?
A) Escalates the agent directly to API
B) Prints the reason provided during the handoff
C) Stores the reason in a database
D) Terminates the agent process
Answer: B

Q5. In the handoff definition, which parameter specifies the type of structured data that the handoff expects?
A) agent
B) on_handoff
C) input_type
D) name
Answer: C

Q6. What does the following code line do?
agent = Agent(name="Escalation agent")
A) Creates a handoff object
B) Initializes a new agent with the name "Escalation agent"
C) Defines the input type for escalation
D) Starts the event loop for async execution
Answer: B

Q1. What happens when a handoff occurs between agents?
A) The conversation history is erased
B) The new agent takes over and can see the entire previous conversation history
C) Only the latest user message is visible to the new agent
D) The input is converted into JSON format
Answer: B

Q2. What is the purpose of an *input_filter* in a handoff?
A) To log the reason for escalation
B) To validate agent responses before execution
C) To modify or filter the conversation history passed to the new agent
D) To terminate unused agents
Answer: C

Q3. Which object type does an input filter function receive and return?
A) BaseModel
B) HandoffInputData
C) RunContextWrapper
D) Agent
Answer: B

Q4. Which module provides common pre-built input filter patterns like removing tool calls from history?
A) agents.utils
B) agents.base
C) agents.extensions.handoff_filters
D) agents.logs
Answer: C

Q5. In the given example, what does handoff_filters.remove_all_tools do?
A) Removes all conversation text
B) Removes all tool call entries from the conversation history
C) Removes the agent configuration
D) Deletes system prompts
Answer: B

Q6. What does the following code snippet create?
agent = Agent(name="FAQ agent")
A) A new handoff filter
B) A handoff object for FAQ agent
C) An agent instance named "FAQ agent"
D) A conversation history logger
Answer: C

Tools in Agent SDK
Q1. What is the main purpose of *tools* in the Agent SDK?
A) To decorate agent names
B) To allow agents to take actions such as fetching data, running code, or calling APIs
C) To store previous chat history
D) To restrict agent access
Answer: B

Q2. Which of the following is **NOT** a class of tools in the Agent SDK?
A) Hosted tools
B) Function calling
C) Agents as tools
D) API keys as tools
Answer: D

Q3. What is the role of *hosted tools*?
A) Run on local machines only
B) Run on LLM servers alongside AI models
C) Only work for image generation
D) Only filter inputs before handoff
Answer: B

Q4. Which tool allows an agent to perform **web searches**?
A) FileSearchTool
B) WebSearchTool
C) ImageGenerationTool
D) LocalShellTool
Answer: B

Q5. What does the **FileSearchTool** do?
A) Retrieves files from your local PC
B) Retrieves information from OpenAI Vector Stores
C) Uploads files to external APIs
D) Converts files into JSON format
Answer: B

Q6. Which hosted tool allows automating computer usage tasks?
A) CodeInterpreterTool
B) ComputerTool
C) LocalShellTool
D) HostedMCPTool
Answer: B

Q7. If you want an agent to execute Python code in a sandboxed environment, which tool should you use?
A) WebSearchTool
B) CodeInterpreterTool
C) LocalShellTool
D) ImageGenerationTool
Answer: B

Q8. Which hosted tool is specifically designed for **image generation** from prompts?
A) FileSearchTool
B) ImageGenerationTool
C) HostedMCPTool
D) LocalShellTool
Answer: B

Q9. Which tool allows running shell commands on your local machine?
A) LocalShellTool
B) FileSearchTool
C) ComputerTool
D) WebSearchTool
Answer: A

Q10. In the given example, the FileSearchTool is initialized with which parameter(s)?
A) query only
B) vector_store_ids and max_num_results
C) tool_name and description
D) url and headers
Answer: B

MCQs on Function Tools

Q1. What allows any Python function to be used as a tool in the Agent SDK?
A) FunctionTool class
B) function_tool decorator
C) RunContextWrapper
D) inspect module
Answer: B

Q2. By default, what is the **name of the tool** when a Python function is used as a tool?
A) The first line of the docstring
B) The name of the Python function
C) A system-generated UUID
D) The value of name_override only
Answer: B

Q3. Where is the **tool description** taken from?
A) Function name
B) The docstring of the function
C) Input schema
D) JSON schema definition
Answer: B

Q4. What automatically creates the schema for function inputs?
A) Python typing hints
B) pydantic
C) inspect + griffe + pydantic
D) json module
Answer: C

Q5. In the example, what does the function fetch_weather return?
A) Temperature value
B) Location coordinates
C) "sunny"
D) Weather API response
Answer: C

Q6. Which decorator argument allows overriding the default tool name?
A) tool_name
B) override_name
C) name_override
D) tool_alias
Answer: C

Q7. What is the purpose of RunContextWrapper in the function read_file?
A) It wraps the agent response into JSON
B) Provides execution context when running tools
C) Converts schema to Pydantic model
D) Disables tool logging
Answer: B

Q8. In the given example, what tools are added to the agent named **Assistant**?
A) Only fetch_weather
B) Only read_file
C) fetch_weather and read_file
D) None
Answer: C

Q9. What Python module is used to inspect the function signature?
A) griffe
B) inspect
C) pydantic
D) typing_extensions
Answer: B

Q10. Which module is used to parse docstrings in order to extract descriptions?
A) typing
B) inspect
C) griffe
D) json
Answer: C

MCQs on Context Management

Q1. In the Agent SDK, the term **context** can refer to:
A) Only data available to LLMs
B) Only data available locally in Python functions
C) Both local data and data available to LLMs
D) The agent's memory storage only
Answer: C

Q2. What class is used to represent **local context** in the Agent SDK?
A) RunContext
B) RunContextWrapper
C) ContextManager
D) LocalContext
Answer: B

Q3. How do you typically provide a custom context object when running an agent?
A) By editing the agent's config file
B) By passing it into Runner.run(..., context=your_object)
C) By overriding the LLM prompt
D) By storing it in a database
Answer: B

Q4. What does the generic type parameter **T** in RunContextWrapper[T] represent?
A) The agent's lifecycle state
B) The schema for function tools
C) The type of your custom context object
D) The token length of conversation
Answer: C

Q5. Which of the following is a key rule about context types during an agent run?
A) Each tool function can use a different context type
B) Every agent, tool, and lifecycle hook must use the same context type
C) Context type is optional and can be ignored
D) Context type is automatically determined by LLM
Answer: B

Q6. What can **local context** typically contain?
A) User IDs, loggers, helper functions, dependencies
B) Only the last user message
C) Only tool schemas
D) Only conversation memory
Answer: A

Q7. Which type of context does the **LLM see** when generating responses?
A) Local context in RunContextWrapper
B) Context available to LLMs
C) Logger objects and helper functions
D) User IDs stored in Python
Answer: B

Q8. Why would you use a dataclass or Pydantic object for context?
A) To reduce memory usage
B) To structure and validate the context data
C) To automatically create prompts
D) To enable image generation
Answer: B

Q9. What is one example of using context in a run?
A) Storing API key inside the LLM prompt
B) Providing user information like username/uid
C) Filtering tool calls
D) Limiting output token size
Answer: B

Q10. Which property allows you to access your context object inside tool functions or lifecycle hooks?
A) .local
B) .context
C) .data
D) .state
Answer: B

MCQs on Agent/LLM Context

Q1. When an LLM is called, what data can it directly see?
A) Local context passed via Python objects
B) Only the conversation history
C) The entire system environment variables
D) All available databases by default
Answer: B

Q2. If you want new data to be available to the LLM, how must it be provided?
A) By storing it in local context only
B) By embedding it in system hardware
C) By making it part of the conversation history
D) By saving it in logs
Answer: C

Q3. What is another name for **Agent instructions** provided to an LLM?
A) User prompts
B) System prompts / Developer messages
C) Tool calls
D) Context wrappers
Answer: B

Q4. What is a common example of data always useful in a **system prompt**?
A) Tool call history
B) User's name or current date
C) JSON schema definition
D) Logger objects
Answer: B

Q5. How is adding context to **Runner.run input** different from system prompts?
A) It is ignored by the LLM
B) It allows messages to appear lower in the hierarchy of conversation history
C) It automatically updates conversation memory
D) It bypasses the LLM filters
Answer: B

Q6. Which approach lets the LLM decide when it needs some data by calling a tool?
A) System prompts
B) Runner.run input messages
C) Function tools
D) Retrieval
Answer: C

Q7. Which methods can provide **on-demand contextual data**?
A) Function tools, retrieval, and web search
B) System prompts only
C) JSON schema injection
D) Agent lifecycle hooks only
Answer: A

Q8. What is the purpose of **retrieval or web search** tools in LLM context?
A) To filter tool usage
B) To ground responses in relevant contextual data
C) To replace system prompts
D) To generate function schemas
Answer: B

Q9. Which method is best suited for **static or always-useful information** like username?
A) Retrieval
B) Function tools
C) System prompt (agent instructions)
D) Runner.run input
Answer: C

Q10. Which method is best suited for fetching **external data from files, databases, or the web**?
A) System prompts
B) Function tools
C) Retrieval and Web search
D) Runner.run input
Answer: C

Q1. Which type of context is visible only to **your Python code** (e.g., tools, hooks)?
A) Local context
B) Agent/LLM context
C) System prompt
D) Retrieval context
Answer: A

Q2. Which type of context is visible to the **LLM during response generation**?
A) Local context
B) Agent/LLM context
C) ContextWrapper only
D) Logger objects
Answer: B

Q3. Which class is used to handle **local context**?
A) ContextWrapper
B) RunContextWrapper
C) FunctionTool
D) RetrievalTool
Answer: B

Q4. If you want to pass a **user ID or logger object** into tools and hooks, where should it go?
A) Agent/LLM context
B) Local context
C) System prompt
D) Conversation history
Answer: B

Q5. If you want the LLM to know the **user's name or current date**, where should you put it?
A) Local context
B) Logger object
C) Agent instructions / system prompt (LLM context)
D) Function wrapper
Answer: C

Q6. Which context must be passed consistently as the **same type** across tools, hooks, and agent run?
A) Agent/LLM context
B) Local context
C) Retrieval data
D) Web search
Answer: B

Q7. Which context strategy lets the LLM **decide when to fetch data** (e.g., weather, files)?
A) Local context
B) System prompts
C) Function tools (LLM context)
D) Runner.run context injection
Answer: C

Q8. To ground an LLM's response with external facts from **files or the web**, which context method is best?
A) Local context
B) Retrieval or Web Search (LLM context)
C) Dataclass context injection
D) Logger functions
Answer: B

Q9. Which context is **not directly visible to the LLM** unless explicitly added to conversation history?
A) Agent/LLM context
B) Local context
C) Retrieval context
D) System prompt
Answer: B

Q10. Which is the **key difference** between local context and LLM context?
A) Local context is for Python execution & dependencies, while LLM context is what the model sees in conversation history
B) Local context is only for JSON schemas, while LLM context is for API calls
C) Local context controls tokens, LLM context controls memory
D) Both are identical
Answer: A

MCQs on Streaming

Q1. What is the purpose of **streaming** in the Agent SDK?
A) To reduce memory usage
B) To provide updates and partial responses during an agent run
C) To save logs of conversation history
D) To batch tool calls
Answer: B

Q2. Which method is used to start a **streamed run** of an agent?
A) Runner.run()
B) Runner.start_stream()
C) Runner.run_streamed()
D) Runner.stream()
Answer: C

Q3. What does Runner.run_streamed() return?
A) StreamEvent object
B) RunResultStreaming object
C) ResponseTextDeltaEvent object
D) Agent object
Answer: B

Q4. Which method allows you to iterate over events in a streamed run?
A) result.get_events()
B) result.async_loop()
C) result.stream_events()
D) result.output()
Answer: C

Q5. What type of events are passed directly from the LLM in OpenAI Responses API format?
A) StreamResultEvent
B) ResponseLogEvent
C) RawResponsesStreamEvent
D) RunResultEvent
Answer: C

Q6. Which event type would you check to stream token-by-token text output from the LLM?
A) response.text
B) response.created
C) response.output_text.delta
D) response.complete
Answer: C

Q7. In the example, which class is used to identify streaming delta text events?
A) StreamEvent
B) ResponseTextDeltaEvent
C) RawResponsesStreamEvent
D) RunContextWrapper
Answer: B

Q8. Why are raw response events useful?
A) They allow saving logs of tool calls
B) They help stream messages to users as soon as generated
C) They allow debugging schema validation
D) They reduce token cost
Answer: B

Q9. In the example code, what agent name is given?
A) Assistant
B) Joker
C) Helper
D) Streamer
Answer: B

Q10. Which Python feature is used to consume stream events asynchronously?
A) await for
B) async for
C) for event in async
D) stream.await()
Answer: B

Q1. What do **RunItemStreamEvents** represent?
A) Token-by-token deltas from the LLM
B) High-level events when an item has been fully generated
C) Errors in function tool execution
D) Only raw responses from the API
Answer: B

Q2. What is the main benefit of RunItemStreamEvents compared to raw response events?
A) They reduce token usage
B) They allow pushing progress updates at the level of complete items (messages, tool calls) instead of per-token updates
C) They only track errors in tools
D) They automatically generate agent instructions
Answer: B

Q3. Which event informs you when the **current agent changes** (e.g., due to a handoff)?
A) RunItemStreamEvent
B) RawResponseStreamEvent
C) AgentUpdatedStreamEvent
D) ItemHelpersEvent
Answer: C

Q4. In the example, which function tool is defined?
A) fetch_weather
B) read_file
C) how_many_jokes
D) count_tokens
Answer: C

Q5. What does the function how_many_jokes() return?
A) Always 5 jokes
B) A random integer between 1 and 10
C) A string describing jokes
D) Nothing (void)
Answer: B

Q6. Which event type is ignored in the example code loop?
A) run_item_stream_event
B) agent_updated_stream_event
C) raw_response_event
D) message_output_item
Answer: C

Q7. What happens when a **tool is called** in the event loop?
A) It is ignored
B) Prints -- Tool was called
C) Prints the system prompt
D) Terminates the run
Answer: B

Q8. What output is shown when a **tool_call_output_item** event is received?
A) -- Tool was called
B) -- Tool output: <output>
C) -- Message output
D) Agent updated: <agent>
Answer: B

Q9. Which helper is used to format and print message outputs?
A) ContextHelpers
B) AgentHelpers
C) ItemHelpers
D) StreamHelpers
Answer: C

Q10. What is printed at the start and end of the run in the example?
A) "=== Run initiated ===" and "=== Run finished ==="
B) "=== Run starting ===" and "=== Run complete ==="
C) "Agent updated" and "Message output"
D) Nothing is printed
Answer: B

MCQs on Guardrails

Q1. What is the main purpose of **guardrails** in agents?
A) To generate faster responses
B) To run checks and validations on inputs/outputs
C) To improve conversation history memory
D) To store agent logs
Answer: B

Q2. Why might you run a guardrail with a **fast/cheap model** before the main agent runs?
A) To increase conversation history length
B) To prevent malicious or unwanted usage before running the expensive model
C) To generate partial responses for the user
D) To bypass system prompts
Answer: B

Q3. What happens if a guardrail detects malicious input?
A) It automatically generates a system prompt
B) It raises an error and stops the expensive model from running
C) It still lets the agent run but logs a warning
D) It changes the input silently
Answer: B

Q4. Which type of guardrail checks the **initial user input**?
A) System guardrail
B) Input guardrail
C) Output guardrail
D) Context guardrail
Answer: B

Q5. Which type of guardrail checks the **final agent output**?
A) Context guardrail
B) Input guardrail
C) Output guardrail
D) Token guardrail
Answer: C

Q6. Using guardrails can help save time and money because:
A) They reduce token size automatically
B) They can block malicious or irrelevant requests before expensive processing happens
C) They shorten the agent's prompt history
D) They compress LLM responses
Answer: B

MCQs on Input Guardrails
Q1. What is the **first step** when an input guardrail runs?
A) It checks if .tripwire_triggered is true
B) It raises an InputGuardrailTripwireTriggered exception
C) It receives the same input passed to the agent
D) It generates the final agent output
Answer: C

Q2. What does the guardrail function produce after running?
A) InputGuardrailTripwireTriggered
B) GuardrailFunctionOutput
C) InputGuardrailResult only
D) A system prompt
Answer: B

Q3. The GuardrailFunctionOutput is wrapped into:
A) OutputGuardrailResult
B) GuardrailWrapperObject
C) InputGuardrailResult
D) RunContextWrapper
Answer: C

Q4. What happens if .tripwire_triggered is **true**?
A) The input is ignored and passed to the agent
B) An InputGuardrailTripwireTriggered exception is raised
C) A system prompt is generated automatically
D) The guardrail function re-runs
Answer: B

Q5. Why are guardrails a property of the **Agent** and not passed to Runner.run()?
A) For performance optimization
B) Because they are always reused across all agents
C) Because guardrails are tied to specific agents, and colocating the code improves readability
D) Because Runner cannot handle guardrail functions
Answer: C

Q6. When do an agent's guardrails run?
A) Only if the agent is the **first agent**
B) Always, regardless of agent position
C) Only after an output is generated
D) Only if explicitly enabled in Runner.run()
Answer: A

MCQs on Output Guardrails

Q1. What is the **first step** in running an output guardrail?
A) Check if .tripwire_triggered is true
B) Receive the output produced by the agent
C) Raise an OutputGuardrailTripwireTriggered exception
D) Wrap the output in an OutputGuardrailResult
Answer: B

Q2. What does the guardrail function produce after running?
A) InputGuardrailResult
B) GuardrailFunctionOutput
C) System Prompt
D) GuardrailWrapper
Answer: B

Q3. The GuardrailFunctionOutput in output guardrails is wrapped into:
A) InputGuardrailResult
B) OutputGuardrailResult
C) RunResultWrapper
D) TripwireResponse
Answer: B

Q4. What happens if .tripwire_triggered is **true** in output guardrails?
A) The agent ignores the output and retries
B) An OutputGuardrailTripwireTriggered exception is raised
C) The result is silently dropped
D) The guardrail function runs again
Answer: B

Q5. When do output guardrails run for an agent?
A) Only if the agent is the **last agent**
B) Only if explicitly set in Runner.run()
C) For every intermediate agent
D) Only when input fails validation
Answer: A

Q6. Why are output guardrails associated with the **Agent** instead of Runner.run()?
A) For better debugging
B) Because guardrails are specific to agents, and colocating them improves readability
C) Because Runner cannot execute guardrails directly
D) To ensure global consistency across all agents
Answer: B

MCQs on Tracing

Q1. What does tracing in the Agents SDK do?
A) Only logs errors during execution
B) Collects a record of events like LLM generations, tool calls, guardrails, etc.
C) Stores user credentials securely
D) Disables guardrails during debugging
Answer: B

Q2. Which of the following is **NOT** included in tracing?
A) LLM generations
B) Tool calls
C) Handoffs
D) Database indexing
Answer: D

Q3. What tool is provided to debug, visualize, and monitor workflows using tracing?
A) Agents Monitor
B) Traces Dashboard
C) OpenAI Debugger
D) Guardrail Console
Answer: B

Q4. Is tracing enabled or disabled by default?
A) Disabled
B) Enabled
C) Optional during installation
D) Enabled only in production
Answer: B

Q5. How can tracing be **globally disabled**?
A) By deleting the agent configuration file
B) By setting the env var OPENAI_AGENTS_DISABLE_TRACING=1
C) By turning off guardrails
D) By removing tool integrations
Answer: B

Q6. How can tracing be disabled for a **single run**?
A) Set agents.run.RunConfig.tracing_disabled to True
B) Use Runner.disable_tracing()
C) Remove tracing logs manually
D) Pass disable=True in Runner.run()
Answer: A

MCQs on Traces and Spans

Q1. What do **Traces** represent in the Agents SDK?
A) A collection of guardrail checks
B) A single end-to-end operation of a workflow
C) Only tool call events
D) A set of user inputs
Answer: B

Q2. What are **Traces** composed of?
A) Tools
B) Guardrails
C) Spans
D) Events
Answer: C

Q3. Which of the following is a property of a Trace?
A) span_data
B) parent_id
C) workflow_name
D) started_at
Answer: C

Q4. What is the correct format for a trace_id?
A) trace_<UUID>
B) trace_<32_alphanumeric>
C) trace_<timestamp>
D) trace_<workflow_name>
Answer: B

Q5. What is the purpose of group_id in a Trace?
A) To assign timestamps to spans
B) To link multiple traces from the same conversation
C) To disable recording of a trace
D) To store span-specific data
Answer: B

Q6. Which Trace property, when set to **True**, ensures that the trace will not be recorded?
A) group_id
B) trace_id
C) disabled
D) metadata
Answer: C

Q7. Which Trace property allows attaching additional information to the trace?
A) metadata
B) span_data
C) workflow_name
D) parent_id
Answer: A

Q8. What do **Spans** represent?
A) Operations with a start and end time
B) Metadata about workflows
C) IDs of chat threads
D) LLM tokens generated
Answer: A

Q9. Which property of a Span connects it to its parent?
A) workflow_name
B) span_data
C) parent_id
D) metadata
Answer: C

Q10. What kind of information does span_data hold?
A) Trace metadata only
B) Start and end timestamps
C) Information about the specific operation, like Agent or LLM generation
D) Unique trace identifiers
Answer: C

MCQs on Orchestrating Multiple Agents

Q1. What does **orchestration** refer to in the context of multiple agents?
A) The training of LLMs
B) The flow of agents in an app, including order and decisions
C) The debugging of workflows
D) The monitoring of guardrails
Answer: B

Q2. Which of the following is **NOT** a main way to orchestrate agents?
A) Allowing the LLM to make decisions
B) Orchestrating via code
C) Tracing through spans
D) Mixing both code and LLM-based orchestration
Answer: C

Q3. How does **LLM-based orchestration** work?
A) By hardcoding the agent flow in advance
B) By letting the LLM plan, reason, and decide the next steps
C) By monitoring guardrails on agent output
D) By disabling trace recording
Answer: B

Q4. What is the main method when orchestration is done via **code**?
A) The code determines which agent runs and in what order
B) The LLM decides the flow dynamically
C) Guardrails are automatically applied to all agents
D) The workflow is visualized in the trace dashboard
Answer: A

Q5. Can you combine LLM-based and code-based orchestration?
A) No, they are mutually exclusive
B) Yes, you can mix and match both patterns
C) Only if guardrails are disabled
D) Only in tracing mode
Answer: B

Q6. Why would someone mix both orchestration methods?
A) To avoid using traces and spans
B) To balance tradeoffs of flexibility and control
C) To ensure guardrails are never triggered
D) To make tracing mandatory
Answer: B

MCQs on Orchestrating via LLM

Q1. What is an **agent** in the context of LLM orchestration?
A) A tool for tracing workflows
B) An LLM equipped with instructions, tools, and handoffs
C) A span inside a trace
D) A guardrail for monitoring inputs
Answer: B

Q2. Which of the following is an example of a **tool** an agent could use?
A) Trace recording
B) Web search
C) Guardrail validation
D) Span monitoring
Answer: B

Q3. What role do **handoffs** play in LLM orchestration?
A) They prevent malicious input from reaching the agent
B) They delegate tasks to specialized sub-agents
C) They execute code directly on the system
D) They disable tracing for efficiency
Answer: B

Q4. When is the **LLM orchestration pattern** especially useful?
A) When the task is fully structured with predefined steps
B) When the task is open-ended and requires reasoning and planning
C) When guardrails are disabled
D) When only code-based orchestration is allowed
Answer: B

Q5. Which tactic is most important for enabling effective orchestration via LLM?
A) Using only one general-purpose agent for all tasks
B) Disabling evals to avoid complexity
C) Investing in good prompts with clear tool usage guidelines
D) Avoiding iteration to save time
Answer: C

Q6. What does **agent introspection** mean in this context?
A) Tracing spans inside a workflow
B) Allowing the agent to critique and improve itself in a loop
C) Running guardrails on user input
D) Monitoring system logs only
Answer: B

Q7. Why is it better to have **specialized agents** instead of one general-purpose agent?
A) Specialized agents are easier to trace
B) Specialized agents excel in specific tasks and improve reliability
C) General-purpose agents cannot use tools
D) Specialized agents do not require prompts
Answer: B

Q8. What is the benefit of investing in **evals** for agents?
A) It disables unnecessary tracing
B) It trains agents to improve and perform tasks better
C) It prevents the need for specialized agents
D) It ensures input guardrails are always triggered
Answer: B

MCQs on Orchestrating via Code

Q1. What is a key advantage of orchestrating via **code** compared to orchestrating via LLM?
A) It makes tasks more creative
B) It makes tasks more deterministic and predictable
C) It reduces the need for structured outputs
D) It eliminates the need for multiple agents
Answer: B

Q2. Which of the following is an example of **structured output orchestration**?
A) Running multiple agents in parallel
B) Asking an agent to classify a task into categories and choosing the next agent accordingly
C) Running an evaluator in a loop to critique outputs
D) Using handoffs to delegate to specialized sub-agents
Answer: B

Q3. What does **chaining multiple agents** involve?
A) Randomly running agents until the task completes
B) Using structured outputs to classify data
C) Passing the output of one agent as the input of the next
D) Running multiple agents simultaneously
Answer: C

Q4. Which of the following best represents a **chained task decomposition**?
A) Writing a blog post by doing research, creating an outline, drafting, critiquing, and improving
B) Running multiple agents in parallel to perform independent tasks
C) Using one general-purpose agent for all steps
D) Disabling tracing for efficiency
Answer: A

Q5. What is the purpose of running an agent in a **while loop with an evaluator agent**?
A) To ensure the output passes specific criteria before final acceptance
B) To run agents in parallel for faster performance
C) To randomly assign agents tasks until one succeeds
D) To reduce the need for prompts
Answer: A

Q6. Which technique can improve **speed** when tasks don't depend on each other?
A) Using structured outputs
B) Running multiple agents in parallel (e.g., asyncio.gather)
C) Running tasks in a while loop with feedback
D) Chaining agents sequentially
Answer: B

Q7. Which orchestration method is best when you want predictable **performance, speed, and cost**?
A) Orchestrating via LLM
B) Orchestrating via guardrails
C) Orchestrating via code
D) Orchestrating via tracing
Answer: C

MCQs on Models in Agents SDK

Q1. Which model type is **recommended** in the Agents SDK for calling OpenAI APIs?
A) OpenAIChatCompletionsModel
B) OpenAIResponsesModel
C) GPT-5-nano
D) LiteLLM models
Answer: B

Q2. What is the **default OpenAI model** used when no model is specified during Agent initialization?
A) gpt-3.5
B) gpt-4.1
C) gpt-5
D) gpt-5-mini
Answer: B

Q3. How can you set a specific model for all agents that don't explicitly define one?
A) By editing agents/models.py
B) By calling set_default_openai_client
C) By setting the OPENAI_DEFAULT_MODEL environment variable
D) By installing LiteLLM
Answer: C

Q4. What are the **default ModelSettings** applied when using GPT-5 models in the SDK?
A) reasoning.effort="high", verbosity="medium"
B) reasoning.effort="low", verbosity="low"
C) reasoning.effort="minimal", verbosity="minimal"
D) reasoning.effort="medium", verbosity="high"
Answer: B

Q5. Which GPT-5 variants are designed for **lower latency**?
A) gpt-5 and gpt-5-large
B) gpt-5-mini and gpt-5-nano
C) gpt-5-pro and gpt-5-ultra
D) gpt-4.1 and gpt-4.0
Answer: B

Q6. Why does the SDK default to **reasoning.effort="low"** instead of "minimal"?
A) Because "minimal" is not supported by GPT-5
B) Because "minimal" is slower than "low"
C) Because some built-in tools (like file search and image generation) do not support "minimal"
D) Because verbosity must always be set to "low"
Answer: C

Q7. If you pass a **non–GPT-5 model name** without custom model settings, what happens?
A) The system defaults to GPT-4.1
B) The SDK applies GPT-5 default settings
C) The SDK reverts to generic ModelSettings compatible with any model
D) The agent execution fails
Answer: C

Q8. How can you use **non-OpenAI models** with the Agents SDK?
A) By default, no support is available
B) By using LiteLLM integration with litellm/ prefixes
C) By creating custom Python wrappers only
D) By overriding the GPT-5 settings
Answer: B

Q9. Which method allows setting a **global custom model provider** for all agents in a run?
A) Agent.model
B) set_default_openai_client
C) ModelProvider at Runner.run level
D) Disabling tracing
Answer: C

Q10. Why do most examples use the **Chat Completions API** instead of the Responses API?
A) Because Responses API is deprecated
B) Because most LLM providers don't yet support the Responses API
C) Because Chat Completions API is faster than Responses
D) Because GPT-5 only works with Chat Completions
Answer: B

MCQs on Lifecycle Events (Hooks)
Q1. What is the purpose of **lifecycle events (hooks)** in an agent?
A) To increase model reasoning power
B) To observe and respond to events during an agent's execution
C) To replace the default model with GPT-5
D) To optimize latency and cost
Answer: B

Q2. Which property is used to connect lifecycle event handlers to an agent?
A) model_settings
B) hooks
C) events
D) tracing
Answer: B

Q3. Which class should you subclass in order to implement custom lifecycle event hooks?
A) Agent
B) AgentRunner
C) AgentHooks
D) AgentLifecycle
Answer: C

Q4. What should you do after subclassing AgentHooks to handle specific lifecycle events?
A) Override the methods corresponding to the events of interest
B) Call set_default_openai_client()
C) Install the litellm dependency group
D) Change the OPENAI_DEFAULT_MODEL variable
Answer: A

Q5. Which of the following is an example use case for lifecycle hooks?
A) Logging agent events
B) Pre-fetching data when a specific event occurs
C) Both A and B
D) Changing default reasoning effort
Answer: C

MCQs on Structured Model Outputs
Q1. What is the main purpose of **Structured Outputs** in LLMs?
A) To generate responses only in plain text
B) To ensure responses strictly follow a JSON Schema defined by the user
C) To speed up inference latency
D) To allow multiple agents to work in parallel
Answer: B

Q2. Why is **JSON** commonly used for Structured Outputs?
A) Because it is only compatible with Python
B) Because it is the default output of GPT models
C) Because it is one of the most widely used formats for applications to exchange data
D) Because JSON prevents hallucinations automatically
Answer: C

Q3. Which of the following is a **benefit of Structured Outputs**?
A) Reduced API costs
B) Reliable type-safety without extra validation
C) Automatic translation of text to any language
D) Increased reasoning effort
Answer: B

Q4. What does “explicit refusals” mean in the context of Structured Outputs?
A) The model will refuse to output JSON
B) Safety-based model refusals can be detected programmatically
C) The user must explicitly refuse invalid data
D) Structured Outputs block unsafe prompts
Answer: B

Q5. Which libraries are used in **OpenAI SDKs** to define object schemas for structured outputs?
A) TensorFlow (Python) and React (JavaScript)
B) Pydantic (Python) and Zod (JavaScript)
C) NumPy (Python) and Next.js (JavaScript)
D) Flask (Python) and Express (JavaScript)
Answer: B

Q6. Without Structured Outputs, what issue might occur with model responses?
A) They may exceed token limits
B) They may omit required keys or produce invalid enum values
C) They may run in infinite loops
D) They may generate only code snippets
Answer: B

MCQs on StreamedAudioResult
Q1. What does the **StreamedAudioResult** class represent?
A) The final text output of an agent
B) The input handler for audio files
C) The output of a VoicePipeline that streams events and audio data as they're generated
D) A JSON schema validator for structured outputs
Answer: C

Q2. Which method in **StreamedAudioResult** streams events and audio data?
A) __init__
B) run()
C) stream()
D) process()
Answer: C

Q3. What is the return type of the stream() method in **StreamedAudioResult**?
A) str
B) AsyncIterator[VoiceStreamEvent]
C) List[AudioFile]
D) dict
Answer: B

Q4. Which of the following parameters is **NOT** required to create a StreamedAudioResult instance?
A) tts_model
B) tts_settings
C) voice_pipeline_config
D) output_format
Answer: D

Q5. What is the role of tts_model in the __init__ method of StreamedAudioResult?
A) Defines the type of text input accepted
B) Specifies the TTS (Text-to-Speech) model to use
C) Configures event listeners
D) Sets up the API key for OpenAI
Answer: B

Q6. Where is the **StreamedAudioResult** class defined in the codebase?
A) src/agents/voice/result.py
B) src/agents/core/output.py
C) src/agents/models/settings.py
D) src/audio/stream/pipeline.py
Answer: A