MCQs on **Handoffs** in OpenAI Agents SDK

**1. What is the primary purpose of a Handoff in the OpenAI Agents SDK?**

* A) To persist conversation history across runs
* B) To delegate tasks to another agent, especially in systems with specialized sub-agents
* C) To validate inputs using guardrails
* D) To convert an agent into a callable tool  
  **Answer:** B — Handoffs allow an agent to delegate tasks to another agent, useful when sub-agents specialize in specific areas

**2. How are handoffs represented internally to the LLM in the SDK?**

* A) As guardrails
* B) As session tokens
* C) As special tool calls like transfer\_to\_<agent\_name>
* D) As asynchronous events in the Runner  
  **Answer:** C — Internally, handoffs are represented as tools for the LLM with names like transfer\_to\_<agent\_name>

**3. The handoffs parameter when defining an Agent can include:**

* A) Only Agent objects
* B) Only Handoff objects created via handoff()
* C) Both Agent objects and handoff()-generated Handoff objects
* D) Strings referencing agent names  
  **Answer:** C — You can pass either agents directly or use handoff(agent) to create a Handoff object

**4. Which of the following is NOT customizable when using the handoff() function?**

* A) tool\_name\_override
* B) on\_handoff callback
* C) input\_filter
* D) model\_settings  
  **Answer:** D — model\_settings is not part of handoff customization. handoff() lets you override tool name, description, set on\_handoff, input\_filter, and more

**5. What is the role of input\_filter in a Handoff object?**

* A) To convert an agent into a tool
* B) Immediately execute the handoff
* C) Filter or modify inputs passed to the next agent
* D) Prevent guardrails from triggering  
  **Answer:** C — input\_filter lets you filter or modify the input history that is passed to the next agent

**6. What does the on\_handoff callback do?**

* A) Determines how inputs are filtered during handoff
* B) Specifies the description shown in tracing data
* C) Runs custom logic (e.g., fetch data) when a handoff is invoked
* D) Defines guardrail policies for the next agent  
  **Answer:** C — on\_handoff is a callback executed when the handoff is invoked; useful for data fetching or other side effects

**7. What's the default name of the transfer tool when creating a handoff to an agent named “Refund Agent”?**

* A) handoff\_refund\_agent
* B) refund\_agent\_transfer
* C) transfer\_to\_refund\_agent
* D) transfer\_refund\_agent  
  **Answer:** C — The default tool name is transfer\_to\_<agent\_name>, e.g., transfer\_to\_refund\_agent

**8. What does the RECOMMENDED\_PROMPT\_PREFIX include?**

* A) Guardrail instructions only
* B) Instructions for handling handoffs, naming conventions, and system context
* C) Tool usage examples
* D) Default model settings  
  **Answer:** B — It includes system context and instructions to treat handoffs as seamless transfers, without drawing attention to them

**9. What is the effect of wrapping prompts with prompt\_with\_handoff\_instructions()?**

* A) Adds recommended system instructions about handoffs to the prompt
* B) Enforces tool usage in the prompt
* C) Embeds conversation history automatically
* D) Activates guardrail logic  
  **Answer:** A — This helper adds recommended handoff context instructions to a prompt

**10. In what scenarios are handoffs particularly useful?**

* A) Modular systems with triage and specialized sub-agents, like support apps with billing or FAQ agents
* B) Single-agent question-answering only
* C) When you only need structured output via Pydantic
* D) For synchronous vs asynchronous execution  
  **Answer:** A — Handoffs are especially advantageous when modularizing responsibilities across specialized agents, such as triage → billing/refund agents