50 High-Quality MCQs on LangChain LCEL & LangGraph

Instructions: Each question has exactly one correct option. Questions are unique, code- and behavior-focused. No answers are provided.

1. In LCEL, what is the primary purpose of the pipe (|) operator?
A. To merge multiple prompts into one
B. To pass the output of one runnable as input to the next
C. To execute runnables concurrently
D. To store intermediate state globally

2. Which LCEL feature enables composability without immediate execution?
A. Static graph compilation
B. Lazy runnable evaluation
C. Token streaming
D. Prompt caching

3. What happens internally when `.invoke()` is called on an LCEL chain?
A. The chain is converted to JSON
B. All runnables execute in sequence with data propagation
C. The LLM is re-initialized
D. The prompt variables are frozen

4. Which method is recommended for constructing role-based prompts in ChatPromptTemplate?
A. from_template
B. from_prompt
C. from_messages
D. from_schema

5. Why is ChatPromptTemplate preferred over PromptTemplate for chat models?
A. It supports tool calling automatically
B. It enforces message role separation
C. It prevents hallucinations
D. It eliminates prompt variables

6. What is the defining characteristic of RunnableLambda?
A. It wraps an LLM with memory
B. It adapts a Python callable into a runnable
C. It executes shell commands
D. It creates async graphs

7. How does RunnableLambda handle inputs and outputs?
A. Only strings are allowed
B. It enforces JSON schemas strictly
C. It passes Python objects transparently
D. It converts everything to text

8. In an LCEL chain, where is RunnableLambda most commonly used?
A. As a replacement for retrievers
B. For lightweight data transformation or routing
C. To manage long-term memory
D. To call external APIs directly

9. What occurs if a RunnableLambda raises an exception during execution?
A. The chain retries automatically
B. The exception propagates and halts execution

C. The error is silently ignored
D. The output is set to None

10. Which capability allows RunnableLambda to participate in LCEL pipelines?
A. Runnable interface compliance
B. Graph node inheritance
C. Token streaming support
D. Built-in retry logic

11. What is the core purpose of RunnableParallel?
A. To batch prompts for an LLM
B. To execute multiple runnables on the same input simultaneously
C. To merge multiple graphs
D. To create async event loops

12. How are outputs from RunnableParallel returned?
A. As a single concatenated string
B. As a dictionary keyed by runnable name
C. As a list in execution order
D. As separate streams

13. What input does RunnableParallel receive?
A. Separate inputs for each runnable
B. A shared input distributed to all runnables
C. Only dictionary inputs
D. Only LLM outputs

14. Which scenario best suits RunnableParallel usage?
A. Multi-step reasoning chains
B. Fan-out feature extraction pipelines
C. Stateful agent loops
D. Prompt engineering experiments

15. How does RunnableParallel differ from LCEL sequential chaining?
A. It changes prompt formatting
B. It avoids shared inputs
C. It enables concurrent execution paths
D. It enforces strict typing

16. What does LCEL fundamentally abstract away from the developer?
A. LLM configuration
B. Execution orchestration details
C. Prompt design
D. Tool definitions

17. Which LCEL property enables easy refactoring of chains?
A. Global state storage
B. Functional composition
C. Hard-coded execution order
D. Implicit retries

18. How does LCEL handle intermediate outputs between runnables?
A. Via global variables
B. Via implicit return-value passing
C. Via message history
D. Via disk persistence

19. What distinguishes LCEL from traditional imperative LangChain code?

A. It removes Python entirely
B. It emphasizes declarative data flow
C. It enforces async-only execution
D. It eliminates debugging

20. Which LCEL feature simplifies reuse of partial pipelines?
A. Chain inheritance
B. Runnable composition
C. Prompt cloning
D. Memory injection

21. In LangGraph, what replaces linear chain execution?
A. Nested runnables
B. Directed graphs with state
C. Prompt stacks
D. Token queues

22. What is the primary responsibility of a LangGraph node?
A. Rendering UI components
B. Mutating or reading shared state
C. Storing embeddings
D. Managing LLM tokens

23. How is state represented in LangGraph?
A. As global variables
B. As an explicit typed state object
C. As hidden memory buffers
D. As prompt templates

24. How does LangGraph decide the next node to execute?
A. Random selection
B. Conditional edges and state evaluation
C. Execution timestamps
D. Prompt similarity

25. What advantage do graphs provide over chains for agent workflows?
A. Reduced token usage
B. Explicit control flow and branching
C. Faster LLM inference
D. Simpler syntax

26. What does `create_react_agent` abstract away from developers?
A. Prompt engineering
B. Manual ReAct loop implementation
C. Tool definitions
D. LLM selection

27. Which reasoning pattern does create_react_agent implement internally?
A. Tree-of-Thought
B. ReAct
C. Chain-of-Verification
D. Self-Consistency

28. What inputs are mandatory when creating a ReAct agent in LangGraph?
A. Only a prompt template
B. LLM and tool definitions
C. Memory and embeddings
D. Dataset and schema

29. How does create_react_agent manage iterations?
A. Fixed number of steps
B. Until a termination condition is met
C. Until token limit is exceeded
D. Until all tools are called

30. What role does state play in create_react_agent execution?
A. It stores UI configuration
B. It tracks thoughts, actions, and observations
C. It replaces memory modules
D. It formats prompts

31. What problem does create_supervisor_agent primarily address?
A. Prompt optimization
B. Coordination of multiple agents
C. Vector search scaling
D. Token streaming

32. How does a supervisor agent differ from a worker agent?
A. It executes tools directly
B. It controls routing and delegation
C. It replaces the LLM
D. It stores embeddings

33. What architectural pattern best describes a supervisor agent system?
A. Observer pattern
B. Master–worker pattern
C. Singleton pattern
D. Factory pattern

34. How do worker agents typically communicate results to the supervisor?
A. Through shared LangGraph state
B. Via REST APIs
C. Through logs only
D. Via prompt templates

35. What happens when multiple worker agents update state?
A. Updates are ignored
B. State transitions follow graph rules
C. The system crashes
D. Outputs are concatenated

36. How does LangGraph prevent uncontrolled agent execution loops?
A. Token limits
B. Explicit graph termination conditions
C. Prompt constraints
D. Timeouts only

37. What is the supervisor agent’s role in error handling?
A. Ignore worker failures
B. Decide retries, fallback, or termination
C. Automatically fix errors
D. Log errors only

38. How do conditional edges enhance supervisor-agent graphs?
A. By reducing token usage
B. By enabling dynamic routing decisions

C. By enforcing static execution
D. By removing state

39. Why is create_supervisor_agent useful for complex workflows?
A. It simplifies UI rendering
B. It centralizes decision-making logic
C. It eliminates the need for tools
D. It guarantees perfect outputs

40. How does supervisor-based design improve scalability?
A. By sharing prompts
B. By delegating tasks across specialized agents
C. By reducing graph nodes
D. By caching LLM responses

41. How can LCEL be used inside LangGraph nodes?
A. As a replacement for state
B. As node execution logic
C. As a memory backend
D. As a routing engine

42. What benefit does LCEL provide when defining node logic?
A. Automatic retries
B. Clean, composable transformations
C. Graph visualization
D. Token compression

43. How is RunnableLambda commonly embedded in a LangGraph node?
A. As a standalone graph
B. As the node's callable execution function
C. As a memory store
D. As a prompt validator

44. What does LangGraph provide that LCEL alone does not?
A. Prompt templates
B. Explicit stateful control flow
C. Runnable composition
D. Lambda execution

45. How does state mutation differ between LCEL and LangGraph?
A. LCEL mutates global state
B. LangGraph enforces explicit state transitions
C. LCEL stores state implicitly
D. LangGraph avoids state entirely

46. What must be carefully managed when enabling parallelism in LangGraph?
A. Prompt length
B. State consistency
C. Token streaming
D. Tool schemas

47. How does LangGraph support iterative reasoning better than LCEL?
A. By enforcing retries
B. By allowing cycles in the graph
C. By reducing latency
D. By caching outputs

48. Which debugging advantage does LangGraph offer?

A. Automatic answer validation
B. Visibility into state transitions
C. Prompt auto-correction
D. Reduced hallucinations

49. How does create_react_agent leverage LangGraph’s execution model?
A. As a static prompt runner
B. As a state-driven reasoning loop
C. As a batch inference tool
D. As a memory store

50. When combining LCEL with supervisor agents, what is most critical?
A. Prompt verbosity
B. Clear separation of concerns
C. Token minimization
D. Tool count