

Experienced Software Test Engineer(QA) Interview Questions and Answers

Manual testing

Selenium with Java

API with Postman

SQL

Java Programming

Java Coding

Git & GitHub

Jenkins

Jira

Docker

CLI commands

AI for QA

Miscellaneous and Tricky

Real Time Scenario Questions

Manual Testing

Category 1 – Basics of Software Testing

1. What is Software Testing?

Software Testing is the process of evaluating and verifying that a software application or system meets the specified requirements and works as intended. It involves executing the software to identify defects, ensuring it is free from critical bugs, and confirming that it delivers the expected business value.

- The main goals are:
 - **Verification:** Does the software meet the design specifications?
 - **Validation:** Does the software meet the customer's needs?
 - Testing is not only about finding defects; it's also about **preventing defects** and **improving quality**.
-

2. Difference between Verification and Validation.

- **Verification** ensures the product is built according to the requirements and design. It answers, “**Are we building the product right?**”
Example: Reviewing documents, inspecting code.
- **Validation** ensures the product meets the actual needs of the user. It answers, “**Are we building the right product?**”
Example: Testing the application with real-world scenarios.

Key point: Verification is a **static process** (no code execution), while Validation is a **dynamic process** (involves running the code).

3. What is the Software Development Life Cycle (SDLC)?

The SDLC is the process of planning, creating, testing, and deploying software. It defines the stages a project goes through from conception to maintenance.

Phases include:

1. Requirement Gathering
2. Design
3. Development
4. Testing
5. Deployment
6. Maintenance

Example: In an e-commerce project, SDLC starts from collecting requirements (shopping cart, payment gateway) to designing UI, developing code, testing, deploying, and fixing post-release issues.

4. What is the Software Testing Life Cycle (STLC)?

STLC is the sequence of specific activities conducted during the testing process to ensure software quality.

Phases include:

1. Requirement Analysis
2. Test Planning
3. Test Case Development
4. Test Environment Setup
5. Test Execution
6. Test Cycle Closure

Each stage has its deliverables. For example, during Test Planning, we produce the **Test Plan Document**; during Test Execution, we produce **Defect Reports**.

5. What are the different levels of testing?

1. **Unit Testing:** Testing individual components (done by developers).
2. **Integration Testing:** Ensuring different modules work together.
3. **System Testing:** Testing the complete system as a whole.
4. **Acceptance Testing:** Ensuring the system meets user needs (UAT).

Example: In an ATM project, first test the PIN verification function (unit test), then combine it with balance checking (integration test), then test the whole ATM workflow (system test), and finally let users test before go-live (acceptance test).

6. What are the types of testing?

- **Functional Testing:** Validates the business logic and features.
- **Non-Functional Testing:** Checks performance, usability, and reliability.
- **Maintenance Testing:** After production changes, ensure existing features still work (Regression Testing).

Example: Testing a login form for valid/invalid credentials is functional testing; testing the speed of login is performance testing.

7. Difference between Manual Testing and Automation Testing.

- **Manual Testing:** Human testers execute test cases without tools. It's flexible and good for exploratory or usability testing.
 - **Automation Testing:** Uses scripts/tools to run tests automatically; suitable for regression, performance, and repetitive tasks.
-

8. What is a Test Case?

A Test Case is a set of conditions and steps that determine whether a software feature works as expected.

Includes: Test Case ID, description, preconditions, steps, expected result, actual result, status.

Example: Test the “Forgot Password” feature—steps include entering an email, checking if a reset link is received.

9. What is a Test Plan?

A Test Plan is a detailed document outlining the testing strategy, scope, schedule, resources, and objectives.

Key sections:

- Objective
 - Scope (In-Scope/Out-of-Scope)
 - Test strategy
 - Entry/Exit criteria
 - Schedule
 - Risks and mitigation
-

10. What is the difference between Severity and Priority?

- **Severity:** Impact of a defect on the system.
Example: A crash in the payment gateway is **Critical Severity**.
 - **Priority:** Order in which the defect should be fixed.
Example: A spelling mistake on the home page may have low severity but high priority if it impacts branding.
-

Category 2 – Test Documentation & Defect Management

11. What is the purpose of Test Documentation?

Test documentation serves as a **knowledge base** for the testing process, ensuring that testing is structured, traceable, and repeatable.

Its main purposes are:

- To communicate **testing scope, approach, and progress** to stakeholders.
- To provide **evidence** that testing was carried out.
- To help in **knowledge transfer** to new team members.
- To **track coverage** and ensure all requirements are tested.

Examples of Test Documentation:

- Test Plan
- Test Cases
- Test Scenario Documents

- Traceability Matrix
 - Test Summary Report
-

12. What is a Test Scenario?

A Test Scenario is a **high-level description** of what to test. It outlines **what functionality or feature** needs to be validated without going into detailed steps.

- Focuses on **user journeys** rather than step-by-step execution.
- Useful when we want to ensure **end-to-end coverage**.

Example:

For an e-commerce site:

Scenario – *Verify that a user can place an order successfully.*

This scenario will cover login → searching → adding to cart → payment → order confirmation.

13. What is a Test Case? How is it different from a Test Scenario?

A **Test Case** is a detailed set of instructions for executing a specific part of a test scenario.

- **Test Scenario** is **broad** (what to test).
- **Test Case** is **detailed** (how to test).

Example:

Scenario – Verify login functionality.

Test Case – Step 1: Open login page → Step 2: Enter valid username/password → Step 3: Click login →

Expected Result: Redirect to dashboard.

14. What is a Traceability Matrix (RTM) and why is it important?

The **Requirement Traceability Matrix** is a document that maps requirements to test cases.

Purpose:

- Ensure **complete coverage** (no missed requirements).
- Helps in **impact analysis** when requirements change.

Structure:

| Requirement ID | Requirement Description | Test Case ID(s) | Status |

Example:

If Req ID *REQ-101* is “User should reset password,” RTM will link it to all test cases covering reset functionality.

15. What is a Bug/Defect in Software Testing?

A defect (or bug) is a deviation of the software from its expected behavior or requirement.

Example:

Expected: Clicking “Add to Cart” should increase the cart count.

Actual: Clicking does nothing → This is a defect.

Key points:

- **Error:** Human mistake in code or requirement.
 - **Defect:** Deviation found during testing.
 - **Failure:** When the defect shows up in production.
-

16. Explain the Defect Life Cycle.

The **Defect Life Cycle** (Bug Life Cycle) describes the stages a defect goes through from discovery to closure.

Typical stages:

1. **New:** Tester logs the defect.
2. **Assigned:** Defect is assigned to a developer.
3. **Open:** Developer starts working on it.
4. **Fixed:** Developer fixes the defect.
5. **Retest:** Tester retests to verify the fix.
6. **Closed:** Defect is confirmed fixed.
7. **Reopened:** If the defect reappears.

Example:

You find the payment button unresponsive → log it → dev fixes → you retest → if working fine, you close it.

17. What is Defect Severity vs. Priority? Give examples.

- **Severity:** Impact on the system’s functionality.
- **Priority:** Urgency of fixing the defect.

Example:

- A spelling mistake in the company name: *Low severity, High priority* (branding issue).
 - System crash when clicking “Checkout”: *High severity, High priority*.
 - Minor color mismatch in footer: *Low severity, Low priority*.
-

18. How do you write a good Bug Report?

A good bug report should be **clear, concise, and complete**.

Include:

- Bug ID
- Title/Summary
- Steps to Reproduce

- Expected Result
- Actual Result
- Screenshots/Logs
- Environment details
- Severity/Priority

Example:

Title: Payment fails when using PayPal.

Steps: Add item → Checkout → Choose PayPal → Click Pay.

Expected: Payment completes successfully.

Actual: “Transaction Failed” error appears.

19. What are Entry and Exit Criteria in Testing?

- **Entry Criteria:** Conditions that must be met before testing begins.
Example: Test environment ready, test data prepared, build deployed.
- **Exit Criteria:** Conditions that must be met before ending testing.
Example: All planned test cases executed, critical defects fixed, test summary report prepared.

20. What is a Test Summary Report?

A Test Summary Report (TSR) is prepared at the end of the testing phase to summarize:

- Scope of testing
- Test execution results
- Defect details and their status
- Coverage analysis
- Risks and recommendations

Category 3 – Test Design Techniques

21. What are Test Design Techniques?

Test Design Techniques are **systematic approaches** to creating test cases that ensure maximum coverage with minimum effort. They help testers focus on **critical areas**, avoid duplication, and discover more defects.

They are generally divided into:

1. **Black-Box Techniques** – Based on requirements, without looking at the code.
2. **White-Box Techniques** – Based on code structure.
3. **Experience-Based Techniques** – Based on tester’s knowledge and experience.

Example: If a login page requires a password of 8–12 characters, you can use **Boundary Value Analysis** to check passwords of 7, 8, 12, and 13 characters.

22. What is Equivalence Partitioning (EP)? Give an example.

Equivalence Partitioning divides test data into **groups (partitions)** that are expected to be treated the same by the system. If one value in a group works, all should work; if one fails, all fail.

Example:

Suppose a field accepts numbers from 1 to 100.

Partitions:

- Valid: 1 to 100 (e.g., test with 50)
- Invalid: < 1 (e.g., 0) and > 100 (e.g., 101)

Benefit: Reduces the number of test cases without losing coverage.

23. What is Boundary Value Analysis (BVA)? Give an example.

BVA focuses on testing at the **edges of input ranges**, as defects often occur at boundaries.

Example:

If the valid input is 18–60 for age:

Test values: 17 (just below), 18 (minimum), 60 (maximum), 61 (just above).

This ensures that both **lower and upper limits** are tested, along with just-outside values.

24. Explain Decision Table Testing with an example.

Decision Table Testing is used for systems with **complex business rules**. It helps ensure all possible **combinations of inputs and outcomes** are tested.

Example: Loan approval:

Conditions:

1. Credit score > 700
2. Income > ₹30,000

Actions: Approve / Reject

Credit > 700	Income > 30k	Action
Y	Y	Approve
Y	N	Reject
N	Y	Reject
N	N	Reject

25. What is State Transition Testing? Give an example.

State Transition Testing checks how the system behaves when **changing from one state to another** based on events/inputs.

Example: ATM machine:

- State: Card inserted → Enter PIN → Valid PIN → Access account.
- Test invalid PIN thrice → State changes to “Card blocked.”

Used when the software has **finite states** and **rules for transitions**.

26. What is Use Case Testing?

Use Case Testing is based on **user scenarios** describing how the system will be used in real life.

- Focuses on **end-to-end workflows**.
- Helps find gaps in integration points.

Example: E-commerce: A “Place Order” use case will include login, product selection, adding to cart, payment, and confirmation.

27. What is Error Guessing?

Error Guessing relies on the **experience and intuition** of the tester to guess where defects are likely to be.

- Not a formal technique.
- Often used alongside other methods.

Example: A tester may guess that an email field might accept invalid formats like `test@.com` and try those inputs.

28. What is Exploratory Testing?

Exploratory Testing is **simultaneous learning, test design, and execution** without predefined test cases.

- Useful when time is short or documentation is incomplete.
- Requires skilled testers with domain knowledge.

Example: Testing a new chat application without test scripts—exploring features, sending random messages, trying special characters, etc.

29. What is Pairwise Testing?

Pairwise Testing is a **combinatorial technique** where testers create test cases covering all possible **pairs of input parameters**.

- Reduces the total number of test cases while covering important interactions.

Example:

Browser (Chrome, Firefox, Edge) × OS (Windows, Mac, Linux) → Instead of testing all 9 combinations, pairwise covers each pair at least once.

30. What is White-Box Testing? How is it different from Black-Box Testing?

- **White-Box Testing:** Based on code structure, checks internal paths, branches, loops. Done by developers or testers with coding skills. Example: Unit Testing, Code Coverage Testing.
- **Black-Box Testing:** Based on requirements without code knowledge. Example: Functional Testing, UI Testing.

Key difference:

- White-box asks “**How is it built?**”
- Black-box asks “**Does it work as expected?**”

Category 4 – Functional Testing Types

31. What is Functional Testing?

Functional Testing verifies that the software’s features work according to the specified requirements. It focuses on **what the system does** rather than how it does it.

- Done using **black-box techniques**.
- Ensures business logic works correctly.

Example: For a banking app, functional testing ensures that money transfers, balance checks, and account statements work as expected.

32. What is Smoke Testing? Why is it called so?

Smoke Testing is a **preliminary check** to verify that the most critical functionalities of an application work before conducting more detailed testing.

- Also known as **Build Verification Testing (BVT)**.
- Ensures the application is stable enough for further testing.

Example: In an e-commerce app, smoke testing might include logging in, searching for a product, adding to cart, and logging out.

Why called “Smoke Testing”? In hardware testing, if a device smoked after powering on, it failed the initial test. The software analogy means, “If the build fails basic tests, it’s not worth continuing.”

33. What is Sanity Testing? How is it different from Smoke Testing?

Sanity Testing is a **narrow, focused** testing to verify specific bug fixes or minor changes in functionality.

- It checks if the specific part of the application works correctly after changes.

Difference:

- **Smoke Testing:** Broad, shallow, early build check.
- **Sanity Testing:** Narrow, deep check after bug fixes.

Example: If a bug in “password reset” is fixed, sanity testing will focus only on that functionality.

34. What is Regression Testing? Why is it important?

Regression Testing ensures that **new changes** (code updates, bug fixes, enhancements) have not broken existing features.

- It's repeated after every change in the application.
- Often automated for efficiency.

Example: If a payment gateway's UI is updated, regression testing ensures that login, search, cart, and other existing features still work.

35. What is User Acceptance Testing (UAT)?

UAT is the final phase of testing where **end users** or **clients** verify that the software meets their business needs.

- Usually done in a staging environment.
- If UAT passes, the software is approved for production.

Example: For an HR payroll system, HR managers check if salary slips, tax deductions, and leave calculations match actual requirements.

36. What is Integration Testing? What are its types?

Integration Testing verifies that **different modules or systems work together correctly**.

Types:

1. **Big Bang:** All modules integrated at once.
2. **Incremental:** Modules integrated step-by-step.
 - **Top-Down:** Higher-level modules tested first.
 - **Bottom-Up:** Lower-level modules tested first.

Example: In an online travel booking site, integrating the flight search module with the payment and ticket generation modules.

37. What is System Testing?

System Testing validates the **entire application as a whole** in a complete, integrated environment.

- Ensures the software meets **both functional and non-functional requirements**.
- Usually the last step before UAT.

Example: For an e-learning platform, system testing would check login, course enrollment, video streaming, payment, and certificate generation in a single end-to-end flow.

38. What is End-to-End (E2E) Testing? How is it different from System Testing?

E2E Testing checks the **complete workflow** of an application from start to finish, including interactions with external systems.

Difference:

- **System Testing:** Focuses on the application in isolation.
- **E2E Testing:** Includes external dependencies (e.g., third-party APIs).

Example: Booking a movie ticket—including selecting a movie, seat booking, payment via a third-party gateway, receiving a confirmation email.

39. What is Exploratory Testing? How is it different from Ad-hoc Testing?

Exploratory Testing is **simultaneous learning, test design, and execution** without pre-written test cases.

- Done by experienced testers to uncover unexpected defects.

Difference:

- **Exploratory:** Structured with a charter (goal).
- **Ad-hoc:** Completely unstructured and random.

Example: While exploring a social media app, a tester tries unusual profile names or long status updates to see if the system handles them.

40. What is Usability Testing? Why is it important?

Usability Testing evaluates how **easy and intuitive** the application is for end users.

- Focuses on **UI design, navigation, and user satisfaction**.
- Often involves real users providing feedback.

Example: In a food delivery app, testers check if users can easily search for a restaurant, apply filters, and place an order without confusion.

Importance: Good usability increases customer satisfaction and reduces training/support needs.

Category 5 – Non-Functional Testing Types

41. What is Non-Functional Testing? How is it different from Functional Testing?

Non-functional testing checks **how well** a system performs under specific conditions, rather than **what it does**.

- Includes performance, scalability, security, usability, and reliability testing.
- Functional testing verifies **features**, while non-functional testing verifies **attributes** of the system.

Example:

- Functional → "Does the login work?"
 - Non-functional → "Does the login happen within 2 seconds under 1,000 users?"
-

42. What is Performance Testing?

Performance Testing checks the speed, responsiveness, and stability of a system under expected workloads.

- Identifies bottlenecks and ensures performance goals are met.

Example: Testing if a banking site loads the account summary within 3 seconds for 500 concurrent users.

43. What is Load Testing?

Load Testing measures system performance under expected user load to ensure it can handle the planned usage.

Example: Simulating 1,000 users logging in simultaneously to see if the application slows down.

44. What is Stress Testing?

Stress Testing evaluates how the system behaves **beyond its normal capacity** until it breaks.

Example: Sending 5x expected traffic to see how the system fails and recovers.

45. What is Scalability Testing?

Scalability Testing checks whether the system can **scale up or down** to handle increased or decreased workloads.

Example: Increasing the number of transactions in a payment system from 100/sec to 1,000/sec and checking if it scales properly.

46. What is Stability Testing?

Stability Testing (soak testing) checks if the system runs without failure for a long period under a certain load.
Example: Running a live video stream for 72 hours to ensure no memory leaks or crashes.

47. What is Security Testing?

Security Testing ensures the system is protected against unauthorized access, vulnerabilities, and attacks.
Example: Testing if an e-commerce site prevents SQL Injection and Cross-Site Scripting (XSS).

48. What is Compatibility Testing?

Compatibility Testing ensures the application works across different devices, browsers, operating systems, and network environments.
Example: Checking if a travel booking site works on Chrome, Firefox, Safari, and Edge on both Windows and Mac.

49. What is Localization Testing?

Localization Testing verifies that the application is adapted for a specific region or language.
Example: Checking if the date format is DD/MM/YYYY in India and MM/DD/YYYY in the US.

50. What is Accessibility Testing?

Accessibility Testing ensures the application can be used by people with disabilities.
Example: Checking if a website supports screen readers and has proper alt text for images.

Category 6 – Defect Management & Bug Reporting

51. What is Defect Leakage?

Defect Leakage happens when a bug escapes from one testing phase and is found in a later stage.
Example: A bug missed during system testing but found during UAT.

52. What is Defect Density?

Defect Density = Number of confirmed defects / Size of the software module.
It helps measure the quality of code.
Example: 10 defects found in 500 lines of code → Density = 0.02 defects per LOC.

53. What is Defect Clustering?

Defect Clustering means that most defects are found in a small number of modules due to high complexity or poor design.

Example: 80% of bugs being in the payment gateway module.

54. What is a Bug Triage?

Bug Triage is a meeting to prioritize and assign defects based on severity, priority, and business impact.

55. What is a Showstopper Defect?

A showstopper is a critical defect that blocks testing progress.

Example: Application crashes immediately after login.

56. What is a Latent Defect?

A defect that exists but is not detected during testing and appears later under specific conditions.

Example: A bug in a tax calculation system that only appears at year-end.

57. What is a Masked Defect?

A defect hidden by another defect.

Example: A crash in the payment page hides a bug in the discount calculation.

58. What is Root Cause Analysis (RCA)?

RCA identifies the **underlying cause** of a defect to prevent recurrence.

Example: Investigating why a login bug occurred → Found incorrect password hashing logic.

59. What is a Bug Life Cycle Tool? Give examples.

Bug Life Cycle tools manage defects from logging to closure.

Examples: Jira, Bugzilla, Mantis, Azure DevOps.

60. What is the purpose of logging defects early?

Catching bugs early reduces the cost of fixing them, as the cost of a defect increases in later stages of SDLC.

Category 7 – Agile Testing Concepts

61. What is Agile Testing?

Agile Testing is testing that follows Agile principles, where testing and development happen simultaneously.

62. What are User Stories?

User Stories describe features from the end-user perspective.

Example: “As a user, I want to reset my password so that I can regain access to my account.”

63. What is Acceptance Criteria in Agile?

Conditions that a user story must meet to be considered complete.

64. What is Sprint Testing?

Sprint Testing is testing within a sprint cycle, including functional and regression tests.

65. What is Continuous Integration in Agile?

Continuous Integration means developers frequently merge code changes into a shared repository, followed by automated builds and tests.

66. What is Shift-Left Testing?

Shift-Left means starting testing earlier in the SDLC to find bugs sooner.

67. What is a Product Backlog?

A prioritized list of all desired features, enhancements, and fixes.

68. What is a Sprint Retrospective?

A meeting at the end of a sprint to discuss what went well and what to improve.

69. What is a Definition of Done (DoD)?

A checklist that a product increment must meet to be considered complete.

70. What is Pair Testing?

Two testers (or a tester and a developer) work together on the same feature to increase coverage and find more bugs.

Category 8 – Test Metrics & Measurement

71. What are Test Metrics?

Test metrics are measurements that track the effectiveness of testing.

Examples: Test coverage, defect density, test execution rate.

72. What is Test Coverage?

Test coverage measures how much of the application's functionality is tested.

73. What is Requirement Coverage?

Requirement coverage ensures that all requirements have at least one test case linked.

74. What is Code Coverage?

Code coverage measures the percentage of source code executed during testing.

75. What is Test Execution Status?

It shows the percentage of planned tests that are executed and their pass/fail rates.

76. What is Test Case Effectiveness?

Measures the ability of test cases to detect defects.

77. What is Defect Removal Efficiency (DRE)?

$DRE = (\text{Defects found in testing} / \text{Total defects}) \times 100.$

78. What is Test Productivity?

Number of test cases created or executed per tester per day.

79. What is Mean Time to Detect (MTTD)?

Average time taken to detect a defect after it is introduced.

80. What is Mean Time to Repair (MTTR)?

Average time taken to fix a defect after it is reported.

Category 9 – Common Testing Challenges

81. What challenges do you face in Manual Testing?

- Unclear requirements
 - Limited time for testing
 - Frequent changes in requirements
 - Incomplete test data
-

82. How do you handle requirement changes during testing?

By impact analysis, updating test cases, and communicating with stakeholders.

83. How do you test when requirements are unclear?

By exploratory testing, involving business analysts, and validating with prototypes.

84. How do you prioritize test cases?

Based on risk, business impact, and criticality.

85. How do you ensure maximum coverage in less time?

By using risk-based testing, prioritizing core features, and automation for regression.

86. What do you do when a developer disagrees with a bug?

Provide clear reproduction steps, evidence (screenshots/logs), and reference requirements.

87. How do you manage testing under tight deadlines?

Focus on critical features, do smoke tests first, and parallelize execution.

88. How do you test with limited test data?

Use data generation tools or create synthetic test data.

89. How do you report testing progress to management?

Through daily status reports, dashboards, and defect trend charts.

90. How do you ensure quality when there's pressure to release?

Run smoke tests, critical regression tests, and communicate risks clearly.

Category 10 – Miscellaneous Manual Testing Topics

91. What is Ad-hoc Testing?

Unstructured, informal testing without documented test cases.

92. What is Monkey Testing?

Randomly providing input to the system to check stability.

93. What is Alpha Testing?

Testing done by internal teams before releasing to users.

94. What is Beta Testing?

Testing by a limited set of end users before full release.

95. What is Installation Testing?

Verifying that software installs, updates, and uninstalls correctly.

96. What is Recovery Testing?

Checking if the system recovers after a crash or failure.

97. What is Compliance Testing?

Ensuring the software meets regulations and standards.

98. What is Internationalization Testing?

Testing if the application supports multiple languages and cultural formats.

99. What is Configuration Testing?

Verifying the system works with different configurations (hardware, OS, browsers).

100. What is the difference between Static Testing and Dynamic Testing?

- Static Testing: Reviews, walkthroughs, inspections (no code execution).
 - Dynamic Testing: Executing code to find defects.
-

Selenium With Java

Section 1 — Core Selenium Basics

1. What is Selenium, and what are its components?

Selenium is an open-source suite of tools for automating web browsers. It allows QA engineers to simulate user interactions like clicking buttons, entering text, and navigating between pages.

Selenium is not a single tool; it's a suite with four major components:

- **Selenium IDE:** A record-and-playback tool, mainly used for quick automation without coding.
- **Selenium RC (Remote Control):** Older tool, now deprecated; allowed automation in different programming languages.
- **Selenium WebDriver:** The most popular component, allowing direct browser communication without any intermediary.
- **Selenium Grid:** Enables parallel test execution across multiple machines and browsers, improving test execution time.

Why it's important in interviews: You need to show you know the whole suite, not just WebDriver.

2. What are the advantages of Selenium over other automation tools?

- **Open-source and free** — No licensing costs.
 - **Multi-language support** — Works with Java, Python, C#, Ruby, JavaScript, etc.
 - **Cross-browser compatibility** — Supports Chrome, Firefox, Safari, Edge, Opera.
 - **Cross-platform execution** — Works on Windows, macOS, and Linux.
 - **Integration with tools** — Works with Maven, Jenkins, TestNG, Cucumber, Git, etc.
 - **Community support** — Large user base with active forums and GitHub repos.
 - **Real browser execution** — Unlike headless testing tools, it works with real browsers for better reliability.
-

3. Difference between Selenium WebDriver and Selenium RC.

Feature	Selenium RC	Selenium WebDriver
Architecture	Requires server to be up and running	Directly interacts with browser
Speed	Slower due to server communication	Faster execution
API Complexity	More verbose commands	Simpler and cleaner API
Browser Support	Limited	Extensive
Status	Deprecated	Current standard

4. Explain the architecture of Selenium WebDriver.

- **Client Libraries:** Language-specific bindings (Java, Python, C#, etc.).
 - **JSON Wire Protocol / W3C Protocol:** Defines the communication between client libraries and browser drivers.
 - **Browser Drivers:** Each browser has its own driver (ChromeDriver, GeckoDriver, etc.).
 - **Browsers:** Actual browser where automation is executed.
- Flow: **Selenium Script** → **Client Library** → **JSON Wire/W3C Protocol** → **Browser Driver** → **Browser**
-

5. What are Locators in Selenium?

Locators are methods to find elements on a web page.

Types:

1. **id** – Finds element by unique id.
 2. **name** – Finds element by name attribute.
 3. **className** – Finds element by class attribute.
 4. **tagName** – Finds element by HTML tag.
 5. **linkText** – Finds links by exact text.
 6. **partialLinkText** – Finds links by partial text.
 7. **cssSelector** – Finds elements using CSS patterns.
 8. **xpath** – Finds elements using XPath expression.
-

6. What is XPath, and what are its types?

XPath is a query language for locating elements in XML/HTML documents.

- **Absolute XPath:** Starts from the root node (`/html/body/div[1]/input`).
 - **Relative XPath:** Starts from the middle of the DOM (`//input[@id='username']`).
- Best Practice:** Prefer relative XPath for stability.
-

7. Difference between Absolute XPath and Relative XPath.

Feature	Absolute XPath	Relative XPath
Start	Root node	Anywhere in DOM
Syntax	/ at start	// at start
Stability	Less stable (changes break)	More stable
Speed	Slightly faster	Slightly slower

8. Explain the difference between findElement() and findElements().

- **findElement():** Returns the first matching element. Throws `NoSuchElementException` if none found.
- **findElements():** Returns a list of matching elements. Returns empty list if none found.

9. How do you handle dynamic elements in Selenium?

Dynamic elements have changing IDs or attributes. Solutions:

- Use **contains()** or **starts-with()** in XPath.
- Use CSS attribute selectors.
- Use **index-based XPath** only if other attributes aren't stable.

10. What are Waits in Selenium, and what types are there?

Waits handle synchronization issues between script and web page.

1. **Implicit Wait:** Sets a default wait for all elements.
2. **Explicit Wait:** Waits for a specific condition (e.g., element visible).
3. **Fluent Wait:** Similar to explicit wait but with polling interval and exception handling.

11. Difference between Implicit Wait and Explicit Wait.

Feature	Implicit Wait	Explicit Wait
Scope	Applied globally to all elements	Specific to one element
Syntax	<code>driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);</code>	<code>new WebDriverWait(driver, 10).until(ExpectedConditions.visibilityOf(element));</code>
Flexibility	Less flexible	Highly flexible

12. How do you handle popups in Selenium?

Types:

- **Alert popup:** `driver.switchTo().alert().accept()`
- **Confirmation popup:** Accept or dismiss.
- **Prompt popup:** Send text and accept.
- **File upload popup:** Use `sendKeys()` with file path or Robot class.

13. How do you handle multiple browser windows in Selenium?

- Get main window handle: `driver.getWindowHandle()`
 - Get all handles: `driver.getWindowHandles()`
 - Switch: `driver.switchTo().window(handle)`
-

14. How do you handle frames and iframes in Selenium?

- Switch by **index**: `driver.switchTo().frame(0)`
 - Switch by **name/id**: `driver.switchTo().frame("frameName")`
 - Switch by **WebElement**: `driver.switchTo().frame(element)`
 - Switch back: `driver.switchTo().defaultContent()`
-

15. How do you handle dropdowns in Selenium?

- Use **Select class**:

```
Select dropdown = new Select(element);  
dropdown.selectByVisibleText("Option");  
dropdown.selectByValue("value");  
dropdown.selectByIndex(2);
```

16. What is Page Object Model (POM) in Selenium?

POM is a design pattern that creates object repositories for UI elements.

Advantages:

- Code reusability
 - Better readability
 - Easy maintenance
-

17. What is TestNG, and why is it used with Selenium?

TestNG is a testing framework for Java.

Advantages:

- Annotations (@Test, @BeforeClass, etc.)
 - Parallel test execution
 - Data-driven testing
 - HTML reports generation
-

18. How do you capture screenshots in Selenium?

```
File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);  
  
FileUtils.copyFile(src, new File("path/screenshot.png"));
```

19. How do you run Selenium tests in multiple browsers?

- Use WebDriver instances for each browser.
 - Use TestNG @Parameters or DataProvider.
 - Use cross-browser testing platforms like BrowserStack.
-

20. What are some limitations of Selenium?

- Works only for web applications, not desktop apps.
 - No built-in reporting (needs TestNG/Extent Reports).
 - Cannot handle CAPTCHA directly.
 - Requires third-party tools for file uploads in OS popups.
-

Section 2 — Java + OOP Concepts for Selenium

21. Why is Java preferred for Selenium automation?

Java is widely used with Selenium because:

- **Platform independence:** Java code runs on any OS with JVM.
 - **Rich library support:** Built-in libraries for file handling, collections, date/time, etc.
 - **Strong OOP support:** Easier to structure large automation projects.
 - **Multi-threading capabilities:** Works well with parallel execution frameworks.
 - **Integration-friendly:** Compatible with TestNG, Maven, Jenkins, and reporting tools.
 - **Large community:** More tutorials, examples, and Q&A for troubleshooting.
-

22. Explain the main OOP concepts and how they apply in Selenium.

1. **Encapsulation:** Wrapping data and methods into classes (Page Object Model uses this heavily).
2. **Inheritance:** Reusing methods/fields from parent classes (e.g., BaseTest class for common methods).
3. **Polymorphism:** Writing code that works in many forms — overloading methods for different inputs, overriding for different browser actions.
4. **Abstraction:** Hiding implementation details (e.g., creating an interface for browser actions, then implementing it for Chrome, Firefox).

23. What is the difference between == and .equals() in Java?

- **== operator:** Compares **references** (whether two variables point to the same object in memory).
- **.equals() method:** Compares **values** of objects.

Example:

```
String a = new String("Test");
String b = new String("Test");
System.out.println(a == b);           // false (different memory)
System.out.println(a.equals(b));      // true (same value)
```

24. How do you handle exceptions in Java Selenium scripts?

- **Try-Catch block:** Handle known exceptions without breaking execution.

```
try {
    driver.findElement(By.id("username")).click();
} catch (NoSuchElementException e) {
    System.out.println("Element not found");
}
```

- **Throws keyword:** Pass exception handling to the caller.
 - **Finally block:** Executes cleanup code (like closing browsers).
-

25. What is the difference between Checked and Unchecked exceptions?

- **Checked:** Checked at compile time (e.g., IOException, FileNotFoundException).
 - **Unchecked:** Occur at runtime (e.g., NullPointerException, ArrayIndexOutOfBoundsException).
In Selenium, most exceptions (like NoSuchElementException) are unchecked.
-

26. What are Access Modifiers in Java, and why are they important in Selenium framework design?

- **public:** Accessible from anywhere.
- **protected:** Accessible within package + subclasses.
- **default:** Accessible within package only.
- **private:** Accessible within class only.

Importance in Selenium:

Use `private` for WebElement variables, and `public` methods for accessing them to maintain encapsulation.

27. What is a Constructor in Java, and how is it used in Selenium POM?

A constructor is a special method used to initialize objects.

In POM, constructors often initialize web elements with PageFactory:

```
public LoginPage(WebDriver driver) {  
    this.driver = driver;  
    PageFactory.initElements(driver, this);  
}
```

This ensures that all elements are ready when a page object is created.

28. Difference between `static` and `non-static` methods in Java.

- **Static:** Belong to the class, called without creating an object.
 - **Non-static:** Belong to an instance, need an object to call.
- In Selenium:** `static` utility methods (like reading config files) save memory.
-

29. What are Java Collections, and where do you use them in Selenium?

Collections are Java classes for storing groups of objects.

Commonly used:

- **List:** Store multiple web elements found by `findElements()`.
 - **Set:** Store unique data (e.g., window handles).
 - **Map:** Store key-value pairs (e.g., test data mapping).
-

30. How do you read data from an Excel file in Java Selenium?

Using Apache POI:

```
FileInputStream fis = new FileInputStream("data.xlsx");  
XSSFWorkbook wb = new XSSFWorkbook(fis);  
XSSFSheet sheet = wb.getSheet("Sheet1");  
String value = sheet.getRow(0).getCell(0).getStringCellValue();
```

This enables data-driven testing.

31. How do you handle multiple classes in Selenium automation?

- Use **POM** to organize page classes.
- Create **utility classes** for reusable functions.
- Maintain **BaseTest class** for setup/teardown.
- Use **inheritance** to reuse methods.

32. Explain method overloading and overriding in Selenium context.

- **Overloading:** Same method name but different parameters (e.g., `clickElement(WebElement e)` and `clickElement(By locator)`).
 - **Overriding:** Subclass modifies parent method behavior (e.g., overriding a `launchBrowser()` method to open Chrome or Firefox differently).
-

33. What is `this` keyword in Java, and where do you use it in Selenium?

`this` refers to the current object instance.

In POM:

```
public LoginPage(WebDriver driver) {  
    this.driver = driver; // Assign parameter to instance variable  
}
```

34. Explain `final`, `finally`, and `finalize()` in Java.

- **final:** Constant variable or method that cannot be overridden.
 - **finally:** Code block that always executes after try-catch.
 - **finalize():** Method called by Garbage Collector before object destruction.
-

35. How do you use `super` keyword in Selenium?

`super` refers to the parent class.

Example: In a `BaseTest` class having `setup()` method, we can call it in child test class:

```
super.setup();
```

36. What is the difference between `String`, `StringBuffer`, and `StringBuilder` in Java?

- **String:** Immutable — changes create new objects.
 - **StringBuffer:** Mutable, thread-safe.
 - **StringBuilder:** Mutable, faster, not thread-safe.
In Selenium, `StringBuilder` is useful when building dynamic XPath.
-

37. How do you implement Data-Driven Testing in Selenium Java?

- Store test data in Excel, CSV, JSON, or database.
- Use Apache POI or Jackson API to read data.

- Pass data to tests using **TestNG DataProvider**.
-

38. How do you use Interfaces in Selenium?

Selenium WebDriver is itself an **interface**:

```
WebDriver driver = new ChromeDriver();
```

This allows switching between browsers easily.

39. How do you use Abstract Classes in Selenium?

Abstract classes define partial implementation:

```
public abstract class BrowserSetup {  
    abstract void launchBrowser();  
}
```

Child classes implement specific logic.

40. What is the difference between **throw** and **throws** in Java?

- **throw**: Used to explicitly throw an exception.

```
public void test() throws IOException { ... }
```

Section 3 — Selenium Advanced & WebDriver Features

41. What is the Actions class in Selenium, and when do you use it?

The **Actions class** in Selenium is used to handle **complex user interactions** like mouse movements, drag-and-drop, right-clicks, double-clicks, and keyboard events.

It follows the **builder pattern**, meaning you can chain multiple actions and then call `.perform()` to execute them.

Example:

```
j  
Actions actions = new Actions(driver);  
actions.moveToElement(element).click().build().perform();
```

Use cases:

- Hover over menus

- Drag and drop
 - Simulating keyboard shortcuts
-

42. How do you perform Drag and Drop in Selenium?

You can use the Actions class:

```
WebElement source = driver.findElement(By.id("source"));
WebElement target = driver.findElement(By.id("target"));
Actions actions = new Actions(driver);
actions.dragAndDrop(source, target).perform();
```

Alternative: Click-hold-move-release:

```
actions.clickAndHold(source).moveToElement(target).release().perform();
```

43. What is JavaScriptExecutor in Selenium, and why is it used?

JavaScriptExecutor allows you to run JavaScript code inside the browser from Selenium. It is useful when:

- Elements are hidden behind overlays
- Normal WebDriver methods fail due to dynamic rendering

Example:

```
JavaScriptExecutor js = (JavaScriptExecutor) driver;
js.executeScript("arguments[0].click();", element);
```

It can also be used to scroll:

```
js.executeScript("window.scrollTo(0,500)");
```

44. How do you scroll a page in Selenium?

Using JavaScriptExecutor:

```
js.executeScript("window.scrollTo(0, document.body.scrollHeight)");
```

Or scroll to an element:

```
js.executeScript("arguments[0].scrollIntoView(true);", element);
```

45. How do you handle hidden elements in Selenium?

- Use `JavaScriptExecutor` to click.
 - Use `Actions.moveToElement()` to hover and make it visible.
 - Use explicit waits until the element is interactable.
-

46. How do you handle SSL certificate errors in Selenium?

For Chrome:

```
ChromeOptions options = new ChromeOptions();
options.setAcceptInsecureCerts(true);
WebDriver driver = new ChromeDriver(options);
```

For Firefox:

```
FirefoxOptions options = new FirefoxOptions();
options.setAcceptInsecureCerts(true);
```

47. How do you run Selenium tests in headless mode?

Headless mode runs the browser without UI, useful for CI pipelines.

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless");
WebDriver driver = new ChromeDriver(options);
```

48. How do you handle file uploads in Selenium?

If the file input is an `<input type="file">` element:

```
j
driver.findElement(By.id("upload")).sendKeys("/path/file.txt");
```

If it's an OS popup:

- Use **Robot class**
 - Use **AutoIT** (Windows only)
 - Use **Sikuli**
-

49. How do you handle file downloads in Selenium?

For Chrome:

```
HashMap<String, Object> prefs = new HashMap<>();
prefs.put("download.default_directory", "/path");
ChromeOptions options = new ChromeOptions();
options.setExperimentalOption("prefs", prefs);
```

```
WebDriver driver = new ChromeDriver(options);
```

50. How do you handle browser notifications in Selenium?

Disable notifications in Chrome:

```
ChromeOptions options = new ChromeOptions();  
options.addArguments("--disable-notifications");
```

51. How do you run parallel tests in Selenium?

Using **TestNG** parallel execution in `testng.xml`:

```
<suite name="Parallel" parallel="tests" thread-count="2">
```

Or use **Selenium Grid** to run on multiple machines.

52. How do you use Selenium Grid?

1. Start Selenium Hub:
`java -jar selenium-server-standalone.jar -role hub`
 2. Start Nodes:
`java -jar selenium-server-standalone.jar -role node -hub
http://localhost:4444/grid/register`
 3. Run tests with **RemoteWebDriver** pointing to hub.
-

53. What is the difference between `driver.close()` and `driver.quit()`?

- `driver.close()`: Closes the current browser tab.
 - `driver.quit()`: Closes all browser windows and ends WebDriver session.
-

54. How do you capture network traffic in Selenium?

Use **BrowserMob Proxy** or DevTools Protocol in Chrome:

```
DevTools devTools = ((ChromeDriver) driver).getDevTools();  
devTools.createSession();  
devTools.send(Network.enable(Optional.empty(), Optional.empty(), Optional.empty()));
```

55. How do you handle cookies in Selenium?

Add:


```
driver.manage().addCookie(new Cookie("name", "value"));
```

Get:

```
Set<Cookie> cookies = driver.manage().getCookies();
```

Delete:

```
driver.manage().deleteAllCookies();
```

56. How do you maximize and resize browser windows in Selenium?

```
driver.manage().window().maximize();  
driver.manage().window().setSize(new Dimension(1024, 768));
```

57. How do you verify tooltips in Selenium?

- Use `getAttribute("title")` for static tooltips.
 - Use Actions to hover and fetch the tooltip element text.
-

58. How do you take full-page screenshots in Selenium?

Chrome-specific:

```
((ChromeDriver) driver).getFullPageScreenshotAs(OutputType.FILE);
```

For normal screenshots:

```
TakesScreenshot ts = (TakesScreenshot) driver;  
File src = ts.getScreenshotAs(OutputType.FILE);
```

59. How do you handle shadow DOM elements in Selenium?

Shadow DOM elements can't be accessed directly via `findElement()`.

Use `JavaScriptExecutor`:

```
WebElement shadowHost = driver.findElement(By.cssSelector("#shadow-host"));  
WebElement shadowRoot = (WebElement) js.executeScript("return arguments[0].shadowRoot",  
shadowHost);  
shadowRoot.findElement(By.cssSelector("#inside-element")).click();
```

60. How do you handle stale element exceptions in Selenium?

Occurs when DOM changes after finding an element.

Solution:

- Re-locate the element before action
- Use **Explicit Wait**
- Wrap in retry logic

Section 4 — Automation Frameworks & Tools

61. What is an automation framework, and why is it important in Selenium projects?

An **automation framework** is a structured set of guidelines, coding standards, tools, and practices for building and maintaining automated test scripts.

Importance:

- **Code reusability** (common functions in one place)
 - **Maintainability** (easy to update tests when UI changes)
 - **Scalability** (supports more tests without rework)
 - **Reporting** (detailed test results)
 - **Integration** (works with CI/CD tools like Jenkins)
-

62. What are the types of Selenium automation frameworks?

1. **Linear (Record and Playback):** Simple scripts; hard to maintain.
 2. **Modular:** Breaks app into modules; reusable code.
 3. **Data-Driven:** Reads input data from external sources (Excel, CSV, DB).
 4. **Keyword-Driven:** Uses action keywords to represent steps.
 5. **Hybrid:** Combination of data-driven + keyword-driven.
 6. **Behavior-Driven (BDD):** Uses plain English scenarios (Cucumber, JBehave).
-

63. What is TestNG, and why is it preferred over JUnit for Selenium?

TestNG (Test Next Generation) is a Java testing framework.

Why better than JUnit:

- Annotations are easier to use (@BeforeClass, @Test, etc.)
 - Built-in parallel execution
 - HTML reports
 - Flexible test configuration (testng.xml)
 - Supports data providers
-

64. Explain the TestNG annotations used in Selenium.

- **@BeforeSuite / @AfterSuite:** Run before/after entire suite.
- **@BeforeTest / @AfterTest:** Run before/after <test> in XML.
- **@BeforeClass / @AfterClass:** Run before/after test class.
- **@BeforeMethod / @AfterMethod:** Run before/after every test method.
- **@Test:** Marks a method as a test case.

Example:

```
@BeforeMethod
public void setup() { ... }

@Test
public void testLogin() { ... }

@AfterMethod
public void tearDown() { ... }
```

65. How do you implement Data-Driven Testing in TestNG?

Use **@DataProvider**:

```
@DataProvider(name="loginData")
public Object[][] getData() {
    return new Object[][] {{"user1","pass1"}, {"user2","pass2"}};
}

@Test(dataProvider="loginData")
public void loginTest(String user, String pass) { ... }
```

66. How do you run tests in multiple browsers using TestNG?

Use **@Parameters** in testng.xml:

```
<parameter name="browser" value="chrome"/>
```

In code:

```
@Parameters("browser")
@BeforeMethod
public void setup(String browser) {
    if(browser.equals("chrome")) driver = new ChromeDriver();
}
```

67. What is Maven, and how is it used in Selenium projects?

Maven is a **build automation tool** that manages:

- **Dependencies** (Selenium, TestNG jars)
- **Build lifecycle** (compile, test, package)

- **Project structure** (standard directory layout)
Example: Add Selenium dependency in `pom.xml`:

```
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>4.20.0</version>
</dependency>
```

68. How do you create a Page Object Model (POM) in Selenium?

Steps:

1. Create one Java class per page.
2. Define `WebElements` using `@FindBy`.
3. Write methods for interactions.

Example:

```
public class LoginPage {
    @FindBy(id="username") WebElement userField;
    public void enterUsername(String user) { userField.sendKeys(user); }
}
```

69. What is PageFactory in Selenium?

A class in Selenium that **initializes `WebElements`** declared in a page object.

```
PageFactory.initElements(driver, this);
```

Benefits:

- Reduces boilerplate `findElement()` calls.
 - Improves readability.
-

70. What is Cucumber, and why is it used with Selenium?

Cucumber is a **BDD (Behavior-Driven Development)** tool that allows writing tests in plain English using Gherkin syntax.

Benefits:

- Bridges communication between QA, Dev, and Business teams.
- Readable scenarios like:

```
Given I open the login page
When I enter valid credentials
Then I should be logged in
```

71. Explain the structure of a Cucumber project.

- **Feature files:** Contain scenarios in Gherkin.
 - **Step Definitions:** Java methods linked to steps.
 - **Test Runner:** Runs the features using JUnit/TestNG.
-

72. What is Extent Reports, and how do you use it in Selenium?

Extent Reports is a reporting library that produces beautiful HTML reports.

Basic setup:

```
ExtentReports extent = new ExtentReports();
ExtentTest test = extent.createTest("Login Test");
test.pass("Step passed");
extent.flush();
```

73. How do you integrate Selenium tests with Jenkins?

Steps:

1. Install Jenkins.
 2. Create a new job.
 3. Pull project from Git.
 4. Add Maven build step: `mvn clean test`.
 5. Schedule or trigger builds.
-

74. How do you use Git in Selenium projects?

- **Clone repo:** `git clone url`
 - **Commit changes:** `git commit -m "message"`
 - **Push to repo:** `git push origin branch`
Git helps in version controlling test scripts.
-

75. What is CI/CD, and why is it important in automation?

CI/CD stands for **Continuous Integration / Continuous Deployment**.

In automation:

- CI ensures tests run on every code commit.
 - CD ensures tested code gets deployed automatically.
Benefits: Early bug detection, faster releases.
-

76. How do you parameterize tests in Cucumber?

Using Scenario Outline:

```
Scenario Outline: Login Test
Given I open login page
When I enter "<username>" and "<password>"
Then I should see "<message>"
```

Examples:

```
| username | password | message |
| user1    | pass1    | Welcome User1! |
```

77. How do you run Cucumber tests in parallel?

- Use TestNG runner with multiple threads.
 - Use **maven-surefire-plugin** with `parallel=methods`.
-

78. What is the difference between JUnit and TestNG in Selenium?

Feature	JUnit	TestNG
Parallel Execution	No	Yes
Dependency Tests	No	Yes
Annotations	Limited	Rich set
Reports	Basic	HTML reports

79. How do you generate HTML reports in TestNG?

TestNG automatically generates reports in the `test-output` folder.
For enhanced reports, integrate **Extent Reports** or **Allure Reports**.

80. How do you create a Hybrid Framework in Selenium?

A hybrid framework combines **data-driven** + **keyword-driven** approaches.

Steps:

1. Store test data in Excel.
2. Store keywords like "click", "enterText".
3. Use Java reflection to execute corresponding methods.

Section 5 — Real-time Scenarios, Best Practices, and Troubleshooting

81. In your Selenium project, how do you decide what to automate and what not to automate?

Automation is most effective when applied to:

- **Repetitive tasks** (e.g., regression testing)
 - **Stable features** (UI doesn't change often)
 - **Data-driven tests** with multiple inputs
 - **Critical workflows** (login, checkout)
Avoid automating:
 - Features with **frequent UI changes**
 - **Exploratory tests**
 - Tests involving **CAPTCHAs** or **2FA**, unless bypass methods are available.
-

82. How do you handle dynamic waits in a slow-loading web application?

- Use **Explicit Waits** with `ExpectedConditions` instead of `Thread.sleep()`.
- Combine with **Fluent Wait** to retry at intervals.

```
new WebDriverWait(driver, Duration.ofSeconds(20))

.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));
```

This ensures scripts wait only as long as necessary.

83. What would you do if `click()` is not working on a button?

Possible solutions:

- Use `Actions` class to click.
- Use `JavaScriptExecutor`:

```
js.executeScript("arguments[0].click();", element);
```

- Scroll into view before clicking.
 - Ensure element is **not overlapped** by another element.
-

84. How do you automate CAPTCHA in Selenium?

Direct CAPTCHA automation is **not recommended** for security and legal reasons.

Workarounds:

- Use test environments with CAPTCHA disabled.
 - Use pre-defined test keys.
 - Integrate with CAPTCHA-solving APIs (rare in QA, mostly in scraping).
-

85. How do you handle elements inside multiple nested iframes?

Switch to each iframe in sequence:

```
driver.switchTo().frame("frame1");
driver.switchTo().frame("frame2");
// work here
driver.switchTo().defaultContent();
```

Or use `parentFrame()` to go one step back.

86. How do you handle “ElementClickInterceptedException” in Selenium?

- Wait until element is clickable.
 - Scroll into view.
 - Ensure no overlays are covering it.
 - Use JavaScript click if unavoidable.
-

87. What’s your approach to debugging a failed Selenium test?

1. **Check logs** for exceptions.
 2. **Reproduce manually**.
 3. **Add screenshots** for failed steps.
 4. **Verify locator** hasn’t changed.
 5. **Check waits** — might be timing issue.
 6. **Run in debug mode** in IDE.
-

88. How do you avoid flaky tests in Selenium?

- Use **reliable locators** (avoid index-based XPath).
 - Implement **explicit waits** instead of hard sleeps.
 - Avoid dependency on network speed.
 - Reset test data before execution.
-

89. How do you handle a stale element after a page refresh?

- Re-locate the element after refresh.
- Use ExpectedConditions:

```
ExpectedConditions.refreshed(ExpectedConditions.visibilityOf(element));
```

90. How do you capture logs from the browser console?

For Chrome:

```
LogEntries logs = driver.manage().logs().get(LogType.BROWSER);
for (LogEntry entry : logs) {
    System.out.println(entry.getMessage());
}
```

91. How do you run Selenium tests in Docker?

- Use **Selenium Docker images** (selenium/standalone-chrome).
 - Start containers with Docker Compose.
 - Run tests pointing to the container's RemoteWebDriver.
-

92. How do you integrate API testing with Selenium UI tests?

- Use **RestAssured** or **HttpClient** to perform backend validation.
 - Fetch API data, then compare with UI data in Selenium.
-

93. How do you measure page load time in Selenium?

Using JavaScriptExecutor:

```
Long loadTime = (Long) js.executeScript(
    "return performance.timing.loadEventEnd - performance.timing.navigationStart;");
```

94. How do you test responsive design in Selenium?

- Resize browser window with `setSize()`.
 - Use Chrome DevTools emulation.
 - Integrate with mobile emulators.
-

95. How do you take screenshots only on test failure?

In TestNG:

```
@AfterMethod
public void tearDown(ITestResult result) {
    if (ITestResult.FAILURE == result.getStatus()) {
        // take screenshot
    }
}
```

96. How do you avoid running all tests when only one fails in a suite?

- Use TestNG `dependsOnMethods`.
 - Tag tests with `groups` and run specific groups.
-

97. How do you run Selenium tests without installing a browser locally?

- Use **Cloud Selenium Grids** like BrowserStack, SauceLabs, LambdaTest.
 - Run with `RemoteWebDriver` using cloud credentials.
-

98. How do you validate broken links in Selenium?

1. Collect all `<a>` tags.
2. Extract `href`.
3. Send HTTP request and check status code.

Example:

```
URLConnection conn = (URLConnection) new URL(url).openConnection();
conn.setRequestMethod("HEAD");
```

99. How do you test file uploads in remote execution (like Grid or Cloud)?

- Use **RemoteWebDriver's LocalFileDetector**:

```
((RemoteWebDriver) driver).setFileDetector(new LocalFileDetector());
```

100. What's your strategy when Selenium cannot automate a specific step?

- Use API calls instead of UI for that step.
- Use database validation.
- Mock services to bypass the UI step.
- Discuss feasibility with the team — sometimes **manual execution is necessary**.

API Testing with Postman

A. API Basics & Fundamentals

1. What is API testing and why is it important?

API testing is the process of verifying that Application Programming Interfaces work as intended — not just in terms of returning correct data, but also in meeting functional, performance, and security requirements.

Unlike UI testing, where you check the front-end interface, API testing focuses on the business logic layer — ensuring that endpoints correctly process requests and return expected responses.

It's important because:

- APIs are the backbone of most modern applications (especially microservices).
- Issues at the API level can break multiple downstream systems.
- Faster feedback: API tests execute quicker than UI tests.

Example: In an e-commerce app, the **Add to Cart API** must correctly handle stock availability, pricing, and user authentication before the UI can display it.

2. What are the different types of APIs?

- **Public APIs** – Available to any developer (e.g., Google Maps API).
- **Private APIs** – Internal to an organization.
- **Partner APIs** – Shared with specific business partners under agreements.
- **Composite APIs** – Combine multiple APIs into one call (common in microservices).

Knowing API type is important because it influences authentication, security, and testing scope.

3. What is the difference between SOAP and REST APIs?

SOAP (Simple Object Access Protocol)

- Uses XML format for requests/responses.
- Strict contract (WSDL).
- Heavy, more secure by default.
- Suitable for complex enterprise systems.

REST (Representational State Transfer)

- Uses HTTP methods (GET, POST, PUT, DELETE).
- Can return JSON, XML, HTML.
- Lightweight and faster.
- Most popular for modern web/mobile apps.

4. Explain HTTP methods in API testing.

- **GET** – Retrieve data from the server.
- **POST** – Create new resources.
- **PUT** – Update a resource completely.
- **PATCH** – Update a resource partially.
- **DELETE** – Remove a resource.

Example:

GET /users/1 – fetch user with ID 1

PUT /users/1 – update the entire user object

PATCH /users/1 – update only part of the user data

5. What is idempotency in HTTP methods?

Idempotency means sending the same request multiple times will result in the same outcome.

- **GET, PUT, DELETE** → Idempotent.
- **POST** → Not idempotent.

Example:

- PUT /user/1 { "name": "John" } → No matter how many times you send it, result remains the same.
 - POST /user { "name": "John" } → Multiple sends will create multiple users.
-

6. Difference between URI, URL, and Endpoint.

- **URI** – Identifier for a resource (more general).
- **URL** – A type of URI that includes the location (e.g., protocol + domain + path).
- **Endpoint** – Specific URL where an API resource can be accessed.

Example:

- **URI**: /users/123
 - **URL**: https://api.example.com/users/123
 - **Endpoint**: GET request to that URL.
-

7. Categories of HTTP status codes.

- **1xx** – Informational (100 Continue)
- **2xx** – Success (200 OK, 201 Created)
- **3xx** – Redirection (301 Moved Permanently)
- **4xx** – Client Error (400 Bad Request, 401 Unauthorized, 404 Not Found)

- **5xx** – Server Error (500 Internal Server Error)
-

8. What are headers in API requests and responses?

Headers carry metadata — such as authentication tokens, content type, caching instructions.

Examples:

- `Content-Type: application/json`
 - `Authorization: Bearer <token>`
 - `Accept-Language: en-US`
-

9. What is JSON and why is it preferred over XML?

JSON (JavaScript Object Notation) is lightweight, easy to read, and natively supported by JavaScript. Compared to XML:

- Smaller payloads.
 - No closing tags → faster parsing.
 - More readable for humans.
-

10. How do you validate an API response?

Validation checks:

- **Status Code** – Expected 200, 201, etc.
 - **Response Body** – Match expected fields and values.
 - **Headers** – Verify content type, caching.
 - **Schema Validation** – Ensure response matches defined structure.
-

B. Postman Features & Usage

11. What is Postman and why is it widely used in API testing?

Postman is an API development and testing tool that allows testers and developers to send requests to APIs, analyze responses, and automate test scenarios.

Why it's popular:

- **User-friendly UI** – Even beginners can quickly send API requests.
- **Supports multiple protocols** – HTTP, HTTPS, WebSocket, gRPC.
- **Built-in scripting** – Pre-request and test scripts using JavaScript.
- **Automation support** – Newman CLI allows integration with CI/CD.
- **Environment & variable support** – Useful for switching between dev/staging/prod.

Example: Instead of manually writing curl commands, you can test an API with a click in Postman and save it for re-use.

12. Explain the structure of a request in Postman.

A request in Postman typically consists of:

- **Method** (GET, POST, PUT, DELETE, PATCH)
 - **URL/Endpoint** (e.g., `https://api.example.com/users`)
 - **Headers** (e.g., `Content-Type: application/json`)
 - **Body** (Raw JSON, Form-data, GraphQL query, etc.)
 - **Authorization** (Basic, Bearer Token, OAuth, etc.)
 - **Params** (Query parameters like `?page=2&limit=10`)
-

13. What are Postman Collections and why are they important?

A **Collection** is a folder that organizes related API requests into a single group.

Importance:

- Helps keep test cases grouped logically.
- Supports sharing across teams.
- Can be run as a batch for automation.
- Supports documentation generation.

Example: You can create a **User Management** collection containing APIs for creating, updating, deleting, and retrieving users.

14. What are environments in Postman?

Environments in Postman allow you to store and switch between sets of variables for different API contexts (Dev, Staging, Production).

Example:

- **Dev Base URL:** `https://dev.api.example.com`
- **Prod Base URL:** `https://api.example.com`

By changing the environment, you can run the same requests against different servers without editing the requests manually.

15. What are variables in Postman and how do you use them?

Variables store reusable values to make tests dynamic.

Types of variables:

- **Global** – Available everywhere.
- **Collection** – Limited to a collection.

- **Environment** – Limited to an environment.
- **Local** – Limited to a single request execution.

Usage:

```
{{baseUrl}}/users/{{userId}}
```

Here, `baseUrl` and `userId` can be stored in variables.

16. What is the difference between Pre-request Script and Tests in Postman?

- **Pre-request Script:** JavaScript code that runs **before** the request is sent.
Example: Generating a timestamp or token before request execution.
 - **Tests:** JavaScript code that runs **after** the response is received.
Example: Validating that the status code is 200.
-

17. How do you write a basic test script in Postman?

Example: Checking status code and response time:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});
```

18. How do you send authentication credentials in Postman?

Postman supports multiple authentication types:

- **Basic Auth** – Username/password encoded in Base64.
- **Bearer Token** – Token passed in Authorization header.
- **OAuth 1.0 / 2.0** – Secure authentication flows.
- **API Key** – Passed in query params or headers.

Example for Bearer Token:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...
```

19. What is Collection Runner in Postman?

The Collection Runner allows you to run all requests in a collection in sequence.

You can:

- Pass external test data (CSV/JSON).

- Repeat iterations.
- View summary reports.

Example: Running login → add product → checkout flow in one run.

20. What is Newman and why is it used?

Newman is Postman's command-line tool to run collections in automated pipelines.

Why use it:

- Integrates Postman tests into CI/CD (Jenkins, GitLab CI).
- Supports JSON/HTML reports.
- Executes tests without the Postman GUI.

Command:

```
newman run mycollection.json -e myenv.json
```

21. How do you handle dynamic data in Postman tests?

- Use **pm.variables.set()** to set a variable dynamically.
- Store values from one response to use in another request.

Example:

```
let userId = pm.response.json().id;  
pm.environment.set("userId", userId);
```

22. How do you import/export collections and environments in Postman?

- **Export** – Right-click → Export as JSON.
 - **Import** – Click Import → Choose file or link.
- This is crucial for collaboration and version control.
-

23. What is a Monitor in Postman?

A Monitor runs your collection at scheduled intervals and sends alerts if tests fail.

Example: Run API health checks every hour and email results.

24. How do you validate JSON schema in Postman?

You can use the **tv4** or **Ajv** validator inside tests.

```
const schema = {  
  "type": "object",  
  "properties": {  
    "id": { "type": "number" },
```



```
    "name": { "type": "string" }
  },
  "required": ["id", "name"]
};

pm.test("Schema is valid", function () {
  pm.response.to.have.jsonSchema(schema);
});
```

25. How do you pass query parameters in Postman?

You can:

- Add them in the **Params** tab.
- Append to URL manually (?page=1&limit=5).
- Set dynamically with variables.

C. API Testing in Depth

26. How do you approach testing a new API when no documentation is available?

When there's no documentation, you need to **reverse-engineer** or **explore** the API:

1. **Check available code or network logs** (e.g., browser DevTools, mobile proxy tools like Charles/Fiddler).
2. **Explore using Postman** – start with GET requests to known endpoints.
3. **Check response headers and patterns** – may give hints about parameters or content type.
4. **Look at error messages** – sometimes APIs reveal required fields or validation rules.
5. **Talk to developers** – confirm assumptions before deep testing.

Example: In a mobile app, capturing network traffic during login might reveal `/auth/login` with required body parameters.

27. What are the most common API testing validations?

You should validate:

- **Status Code** – Matches expected result (e.g., 200 for success).
 - **Response Body Content** – Expected fields, values, and data types.
 - **JSON/XML Schema** – Structure matches contract.
 - **Headers** – Content-Type, caching, security headers.
 - **Performance** – Response time is within SLA.
 - **Database** – Data changes match request actions.
 - **Idempotency** – Multiple identical requests give the same result (for idempotent methods).
-

28. How do you test API error handling?

Negative testing is as important as positive testing:

- **Invalid inputs** – Missing fields, incorrect data types.
- **Unauthorized access** – Invalid token, missing token.
- **Resource not found** – Requesting an ID that doesn't exist.
- **Method not allowed** – Sending POST instead of GET.
- **Rate limiting** – Sending too many requests.

Example in Postman:

```
pm.test("Returns 404 when user not found", function () {  
    pm.expect(pm.response.code).to.eql(404);  
});
```

29. What is the difference between functional and non-functional API testing?

- **Functional Testing** – Verifies that the API does what it's supposed to (correct inputs produce correct outputs).
- **Non-functional Testing** – Focuses on performance, security, usability, scalability.

Example:

- **Functional:** Does GET /users/1 return the correct user?
 - **Non-functional:** Does GET /users/1 respond within 300ms under 1000 concurrent requests?
-

30. How do you validate data integrity in API testing?

Steps:

1. **Trigger the API** – e.g., POST /users creates a new user.
 2. **Check API Response** – Ensure the returned data matches input.
 3. **Verify Database** – Run SQL query to ensure data is stored correctly.
 4. **Retrieve via GET API** – Ensure consistency between DB and API output.
-

31. How do you test API security in Postman?

Security testing in Postman includes:

- **Authentication** – Ensure endpoints require proper login.
 - **Authorization** – Ensure users can only access permitted resources.
 - **Token Expiry** – Check that expired tokens are rejected.
 - **Data Exposure** – Ensure sensitive data (passwords, credit cards) is never in plain text.
 - **CORS Handling** – Ensure API restricts requests from unknown domains.
-

32. How do you test an API for performance using Postman?

Postman alone isn't a full performance testing tool, but you can:

- Use `pm.expect(pm.response.responseTime).to.be.below(500)` to check speed.
 - Run APIs repeatedly using **Collection Runner** with multiple iterations.
 - For deeper load testing, export Postman collections to **JMeter** or **k6**.
-

33. How do you chain API requests in Postman?

Chaining means **using output from one API as input to another**.

Example:

```
let token = pm.response.json().token;
pm.environment.set("authToken", token);
```

Then in the next request, set:

```
Authorization: Bearer {{authToken}}
```

This is useful for login flows, order placements, etc.

34. How do you test APIs that have dependencies on other services?

- Use **mock servers** when dependent services are unavailable.
- Test with **stubs** for partial functionality.
- Use **contract testing** to validate API interactions without full integration.

Example: If your payment API depends on a banking API, you can mock the bank's success/failure responses to test your service behavior.

35. How do you handle dynamic tokens or session IDs in Postman?

- Capture the token/session ID from login response:

```
let sessionId = pm.response.json().sessionId;
pm.environment.set("sessionId", sessionId);
```

- Use it in headers or params for subsequent requests:

```
Session-Id: {{sessionId}}
```

36. How do you ensure API backward compatibility?

Backward compatibility means old clients still work when API is updated.

Testing steps:

- Run old test cases on the new API version.
 - Verify deprecated fields still work (until officially removed).
 - Ensure default values are returned when new fields are added.
-

37. How do you test API pagination?

Pagination is tested by:

- **Checking correct page size** – e.g., if `limit=10`, ensure only 10 items returned.
 - **Validating total pages** – Matches total data count.
 - **Testing boundary values** – First page, last page, empty page.
-

38. How do you test sorting and filtering in APIs?

- Send API requests with sorting params (`?sort=name&order=asc`).
 - Verify the list is correctly sorted.
 - Apply filters (`?status=active`) and check only matching records are returned.
 - Combine filters and sorting for complex validations.
-

39. How do you validate API contracts?

An API contract defines **expected request/response structure** (fields, types, rules).

You can:

- Use **JSON Schema validation** in Postman.
 - Compare response with OpenAPI/Swagger definitions.
-

40. How do you test APIs that return asynchronous responses?

Some APIs return an immediate “processing” status and require polling:

1. Send initial request → get job ID.
2. Poll status API until job is completed.
3. Validate final output.

Example in Postman:

```
let status = pm.response.json().status;
pm.test("Processing started", function () {
  pm.expect(status).to.eql("in_progress");
});
```

D. Scripting & Automation in Postman

41. What scripting language does Postman use for automation?

Postman uses **JavaScript** for both **Pre-request Scripts** and **Tests** tabs.

- **Pre-request Scripts** → Execute before sending the API request (used for dynamic data creation, signing requests, generating tokens).
- **Tests** → Execute after receiving a response (used for validation, storing variables, chaining).

Example Pre-request Script:

```
javascript
CopyEdit
pm.environment.set("timestamp", Date.now());
```

42. How do you store and use dynamic values across requests?

You can extract values from a response and store them in environment or global variables using:

```
javascript
CopyEdit
let token = pm.response.json().token;
pm.environment.set("authToken", token);
```

Then in another request, use:

```
css
CopyEdit
Authorization: Bearer {{authToken}}
```

This allows chaining requests like login → perform action → logout.

43. How do you run data-driven tests in Postman?

Data-driven testing in Postman is done using **Collection Runner** with a CSV or JSON data file.

Steps:

1. Create test data file (users.csv).
2. Use variables in requests ({{username}}, {{password}}).
3. Run collection with data file in Collection Runner.

Example CSV:

```
username,password
user1,pass123
user2,pass456
```

44. How do you write assertions for status codes and response times?

```
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is less than 500ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});
```

These ensure APIs return correct codes and are within performance SLAs.

45. How do you validate JSON response body in Postman?

```
let jsonData = pm.response.json();
pm.test("Name is John", function () {
  pm.expect(jsonData.name).to.eql("John");
});
```

You can validate:

- Specific fields (`jsonData.id`)
 - Nested objects (`jsonData.address.city`)
 - Arrays (`jsonData.items.length`)
-

46. How do you validate a JSON schema in Postman tests?

```
const schema = {
  "type": "object",
  "properties": {
    "id": { "type": "number" },
    "name": { "type": "string" }
  },
  "required": ["id", "name"]
};

pm.test("Schema is valid", function () {
  pm.response.to.have.jsonSchema(schema);
});
```

This ensures the API response matches the expected contract.

47. How do you log output to Postman Console?

You can debug scripts using:

```
console.log("Token:", pm.environment.get("authToken"));
```

Postman Console is opened with `View → Show Postman Console`.

48. How do you handle loops in Postman scripts?

Postman does not have native looping between requests in one run, but you can:

- Use **Collection Runner** with multiple iterations.
- Use scripts to call `postman.setNextRequest()` for conditional flows.

Example:

```
if(pm.environment.get("retryCount") < 3) {  
  postman.setNextRequest("Check Status");  
}
```

49. How do you call one request from another in Postman?

You can:

- Use `postman.setNextRequest("Request Name")` for chaining.
- Store variables in Pre-request scripts and reuse them.

Example: Login request sets `{{token}}` → used in subsequent APIs.

50. How do you generate random test data in Postman?

```
pm.environment.set("randomEmail", `user${Math.floor(Math.random()*1000)}@test.com`);  
pm.environment.set("randomId", Math.floor(Math.random()*10000));
```

This is useful for avoiding duplicate data issues during test runs.

51. How do you handle conditional testing in Postman scripts?

```
let status = pm.response.code;  
if (status === 200) {  
  pm.test("Valid Response", () => {  
    pm.expect(pm.response.json().status).to.eql("success");  
  });  
} else {  
  pm.test("Error Response", () => {  
    pm.expect(status).to.be.oneOf([400, 404]);  
  });  
}
```

This ensures flexible validation depending on API behavior.

52. How do you automate Postman tests in CI/CD pipelines?

1. Export the Postman collection and environment.
2. Install **Newman** in your CI/CD environment.
3. Add a build step:

```
newman run MyCollection.json -e MyEnv.json --reporters cli,html
```

4. Fail the build if Newman tests fail.
-

53. How do you capture values from headers in Postman?

```
let contentType = pm.response.headers.get("Content-Type");  
pm.environment.set("contentType", contentType);
```

This is useful for APIs that send tokens in headers instead of body.

54. How do you validate arrays in Postman response?

```
let jsonData = pm.response.json();  
pm.test("Array contains 'Apple'", function () {  
    pm.expect(jsonData.items).to.include("Apple");  
});  
pm.test("Array length is 5", function () {  
    pm.expect(jsonData.items.length).to.eql(5);  
});
```

55. How do you run only selected tests in a Postman collection?

- Use **folders** and run them in Collection Runner.
 - Temporarily disable other requests.
 - Use `postman.setNextRequest(null)` to stop execution after certain tests.
-

56. How do you use environment variables in scripts?

```
pm.environment.set("userId", pm.response.json().id);  
let id = pm.environment.get("userId");
```

Environment variables make requests portable between Dev, QA, and Prod.

57. How do you parameterize requests in Postman?

```
{{baseUrl}}/users/{{userId}}
```


These placeholders are replaced by variable values at runtime.

58. How do you run Postman tests headlessly?

Use **Newman CLI**:

```
newman run mycollection.json -e myenv.json --reporters cli,json
```

This is essential for automation and CI/CD.

59. How do you retry failed requests in Postman scripts?

Postman itself doesn't have automatic retries, but you can simulate:

```
if (pm.response.code !== 200) {  
    postman.setNextRequest("Same Request Name");  
}
```

Limit retries with a counter variable.

60. How do you test APIs with file uploads in Postman?

- Use **form-data** in the Body tab.
 - Select file type and attach the file.
 - Validate upload success in response.
-

E. Advanced Testing Scenarios

61. How do you test versioned APIs?

API versioning ensures backward compatibility when new changes are introduced.

Steps:

1. Identify available versions (/v1/, /v2/).
2. Compare responses for the same request in each version.
3. Ensure old clients still function with older versions.
4. Validate deprecation notices in headers or documentation.
5. Ensure new fields are optional and don't break existing responses.

Example:

GET /v1/users/123 returns {name, email}

GET /v2/users/123 returns {name, email, phone} — old apps should still work without phone.

62. How do you test APIs in a microservices architecture?

Challenges:

- APIs may depend on multiple services.
- Failures in one service can cascade.

Testing strategy:

- Test each service independently with mock data.
- Use **contract testing** (e.g., Pact) to verify integrations.
- Test end-to-end workflows to ensure orchestration is correct.

Example: In an e-commerce app, the **Order Service** API depends on **Inventory Service** and **Payment Service**.

63. How do you test APIs that return asynchronous results?

Some APIs process requests in the background and return job IDs.

Steps:

1. Send initial request (POST).
2. Receive `jobId` and `status pending`.
3. Poll status API until completion.
4. Validate final output or error.

Example in Postman:

```
let status = pm.response.json().status;
if (status !== "completed") {
  postman.setNextRequest("Check Job Status");
}
```

64. How do you test APIs with third-party integrations?

Third-party APIs can be slow, expensive, or unavailable in testing.

Solutions:

- Use **mock servers** to simulate their responses.
 - Validate API error handling when the third party fails.
 - Verify that the correct payload is sent to the external API.
-

65. How do you test API rate limiting?

Rate limits prevent abuse.

Steps:

1. Identify rate limit policy (e.g., 100 requests/min).
2. Send rapid requests until limit is hit.

3. Validate correct error response (often 429 Too Many Requests).
4. Check `Retry-After` header for cooldown time.

66. How do you test webhooks in Postman?

Webhooks send data to your endpoint when an event happens.

Testing:

- Use Postman's **Mock Server** as the webhook receiver.
- Send a test event from the service.
- Validate payload and headers received.

67. How do you test GraphQL APIs in Postman?

Steps:

1. Set request type to POST.
2. In Body → select GraphQL.
3. Write query/mutation:

```
query {  
  user(id: "123") {  
    name  
    email  
  }  
}
```

4. Validate that only requested fields are returned.

68. How do you test APIs with file downloads?

- Send request.
- Check `Content-Disposition` header for filename.
- Verify file type (`Content-Type`).
- Save and compare file size/hash to expected.

69. How do you test APIs that require digital signatures?

- Use Pre-request Scripts to generate HMAC or RSA signatures.
- Use `CryptoJS` in Postman to hash request payload.

Example:

```
const hash = CryptoJS.HmacSHA256(pm.request.body.raw, "secretKey");  
pm.environment.set("signature", hash.toString(CryptoJS.enc.Hex));
```

70. How do you mock APIs in Postman?

Postman Mock Server steps:

1. Create a new mock server.
 2. Define sample responses for endpoints.
 3. Use mock URL in place of real API.
This is useful for testing UI before backend is ready.
-

71. How do you handle API request sequencing in Postman?

- Use `postman.setNextRequest()` to define next request order.
 - Store values from previous responses in variables.
 - Example: Login → Create User → Get User → Delete User.
-

72. How do you test APIs that have different behavior for different users?

- Use different authentication tokens for different roles (admin, user, guest).
 - Validate role-based access control (RBAC).
 - Example: Admin can `DELETE /users/123`, normal user cannot.
-

73. How do you test an API migration from monolith to microservices?

- Identify API endpoints before and after migration.
 - Compare responses for the same input.
 - Test for **performance improvements** and **no functional breakage**.
 - Validate integration points with new microservices.
-

74. How do you test an API that returns streaming data?

Streaming APIs (e.g., WebSocket, SSE):

- Connect to the stream in Postman (WebSocket tab).
 - Send a message and validate incoming stream messages.
 - Check message format and frequency.
-

75. How do you test APIs with scheduled jobs?

If API triggers a scheduled task (e.g., nightly report):

- Use a smaller schedule in test environment.
- Trigger job manually if API supports it.
- Validate job status and output.

76. How do you handle flaky APIs in automation?

- Add retry logic in scripts.
- Validate response before proceeding to next test.
- Log failures for further analysis.

Example:

```
if(pm.response.code !== 200) {  
    postman.setNextRequest("Same Request");}
```

77. How do you test localization in APIs?

- Send requests with `Accept-Language` header (`en-US`, `fr-FR`).
 - Validate responses are translated correctly.
 - Check currency and date formats per locale.
-

78. How do you validate API caching behavior?

- Check `Cache-Control` and `ETag` headers.
 - Send repeated requests and confirm whether server returns cached or fresh data.
-

79. How do you test APIs under failure scenarios?

- Simulate server errors (mock 500 response).
 - Simulate network delays/timeouts.
 - Validate application behavior on API failure.
-

80. How do you test gRPC APIs in Postman?

Postman now supports gRPC:

- Import `.proto` file.
- Select method to test.
- Send request with parameters.
- Validate response fields.

F. CI/CD & Integration Testing

81. How do you integrate Postman tests into CI/CD pipelines?

- Export Postman collection & environment as JSON.
- Install **Newman** (Postman CLI).
- In CI/CD (Jenkins, GitHub Actions, Azure DevOps), add a step to run:

```
newman run collection.json -e environment.json
```

- Fail build if any test fails (--bail).
-

82. What is Newman and why is it used?

- CLI tool to run Postman collections without opening the UI.
 - Used in automation, CI/CD, and scheduled jobs.
 - Supports HTML, JSON, and JUnit reports.
-

83. How do you generate HTML reports from Newman?

Install reporter:

```
npm install -g newman-reporter-html
```

Run with:

```
bash
CopyEdit
newman run collection.json -r html
```

84. How do you schedule Postman collections to run automatically?

Options:

- Use **Newman + cron jobs** on a server.
 - Use **Postman Monitors** (cloud execution on a schedule).
 - Integrate with **Jenkins** to run daily/nightly.
-

85. How do you pass dynamic environment variables in CI/CD?

- Use environment JSON with placeholders.
- Override variables with --env-var flag in Newman:

```
newman run collection.json --env-var baseUrl=https://staging.api.com
```

86. How do you run specific folders or requests from a collection in CI/CD?

```
newman run collection.json --folder "User APIs"
```

Useful when only a subset of tests needs to run.

87. How do you integrate Postman with version control (Git)?

- Export collections as JSON and commit to Git.
 - Or use **Postman API** to sync and fetch latest collection before running.
 - Enables collaboration and CI/CD trigger on changes.
-

88. How do you test APIs automatically after deployment?

- Add Newman run step **post-deploy** in CI/CD.
 - Run smoke tests on deployed environment to ensure it's live.
 - Fail pipeline if smoke test fails.
-

89. How do you run Postman collections in parallel?

Newman doesn't support native parallelism, but:

- Use `newman-reporter-htmlextra` with concurrent test runners.
 - Or split collections into multiple smaller ones and run in parallel in CI/CD.
-

90. How do you publish API test results to a dashboard?

- Export Newman results as JUnit XML.
 - Feed XML into CI/CD dashboards (Jenkins, Azure).
 - Or use custom reporters for HTML dashboards.
-

G. Best Practices & Common Pitfalls

91. What are API testing best practices?

- Test positive & negative scenarios.
- Use environment variables instead of hardcoding.
- Validate status codes, response time, and schema.
- Keep collections modular.

92. How do you ensure API tests are maintainable?

- Group related tests in folders.
 - Use variables for base URLs, tokens, IDs.
 - Use scripts for reusable logic.
-

93. What is schema validation and why is it important?

- Ensures response structure matches API spec.
- Prevents unexpected changes from breaking consumers.

Example:

```
pm.test("Schema valid", function() {  
  pm.response.to.have.jsonSchema(schema);  
});
```

94. What's the difference between smoke, regression, and load testing in APIs?

- **Smoke:** Quick check if API works.
 - **Regression:** Check all functionality after changes.
 - **Load:** Test performance under expected traffic.
-

95. What are common mistakes in Postman testing?

- Hardcoding values.
 - Not testing error responses.
 - Ignoring response time.
 - Missing authentication checks.
-

96. How do you handle API security testing in Postman?

- Test without authentication (should fail).
 - Test with expired tokens.
 - SQL injection & XSS payload checks in inputs.
 - Validate HTTPS is enforced.
-

97. How do you manage large Postman collections efficiently?

- Break into smaller collections.
- Use common pre-request & test scripts in globals.

- Version control collections.
-

98. How do you test APIs with complex JSON structures?

- Use `pm.response.json()` to parse.
- Use `lodash` in scripts to navigate deep fields:

```
javascript
CopyEdit
let value = _.get(pm.response.json(), 'data.user.details.email');
```

99. How do you keep Postman environments in sync across teams?

- Export & share environment JSON.
 - Use Postman workspaces for live sync.
 - Keep secrets in separate secure environments.
-

100. If an API test fails in CI/CD, how do you troubleshoot?

1. Check request/response logs from Newman.
 2. Compare environment variables in local vs CI/CD.
 3. Validate authentication & endpoint availability.
 4. Retry manually in Postman to isolate the issue.
-

SQL

1. What is SQL and why is it important for a QA Engineer?

Answer:

SQL (Structured Query Language) is the standard language used to store, manipulate, and retrieve data from relational databases. For a QA Engineer, SQL is crucial because:

- Most applications store data in databases, so verifying that the data is correct after operations (e.g., form submission, updates) is essential.
- It allows you to perform backend testing without relying on the UI.
- It helps in preparing test data for automation or manual testing.
Example: If you want to check whether a newly registered user appears in the database,

```
SELECT * FROM users WHERE email = 'test@example.com';
```

2. What is the difference between `DELETE`, `TRUNCATE`, and `DROP`?

Answer:

- **DELETE** – Removes specific rows from a table based on a `WHERE` clause. The table structure remains. Transaction logs are kept, so you can rollback if needed.

```
DELETE FROM employees WHERE department = 'HR';
```

- **TRUNCATE** – Removes all rows from a table, resets identity counters, and is faster than `DELETE`. You cannot rollback in most databases.

```
TRUNCATE TABLE employees;
```

- **DROP** – Completely removes the table and its structure from the database.

```
DROP TABLE employees;
```

QA relevance: Knowing the difference helps avoid accidental data loss when preparing or cleaning test data.

3. What is a primary key? Can a table have multiple primary keys?

Answer:

A **primary key** is a column (or set of columns) that uniquely identifies each row in a table and does not allow `NULL` values.

Example:

```
CREATE TABLE students (  
    student_id INT PRIMARY KEY,
```

```
name VARCHAR(50)
);
```

- A table can have **only one primary key constraint**, but that key can be a **composite key** (multiple columns).
QA use case: When inserting test data, you must ensure primary key values are unique to avoid errors.
-

4. What is a foreign key and why is it important in testing?

Answer:

A **foreign key** is a column in one table that refers to the primary key in another table, establishing a relationship between tables.

Example:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Importance for QA:

- Ensures **referential integrity** (you cannot have an order without a valid customer).
 - While testing, you must insert data in the correct order (parent first, then child).
-

5. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN?

Answer:

- **INNER JOIN** – Returns only matching rows from both tables.
- **LEFT JOIN** – Returns all rows from the left table, and matching rows from the right table.
- **RIGHT JOIN** – Returns all rows from the right table, and matching rows from the left table.
- **FULL JOIN** – Returns all rows from both tables, whether or not there's a match.

Example:

```
SELECT c.customer_name, o.order_id
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id;
```

QA use case: Often used in data verification across multiple related tables.

6. How do you find duplicate records in a table?

Answer:

```
SELECT email, COUNT(*)
FROM users
GROUP BY email
HAVING COUNT(*) > 1;
```

Explanation:

- GROUP BY groups records by email.
 - HAVING COUNT(*) > 1 filters only those emails that appear more than once.
- QA use case:** To validate if the application prevents duplicate entries.
-

7. What is the difference between WHERE and HAVING?

Answer:

- **WHERE** – Filters rows before grouping.
- **HAVING** – Filters groups after aggregation.

Example:

```
-- Using WHERE
SELECT * FROM employees WHERE department = 'IT';

-- Using HAVING
SELECT department, COUNT(*)
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

QA note: When validating aggregated reports, you'll often use HAVING.

8. How do you find the second highest salary from a table?

Answer:

```
SELECT MAX(salary) AS second_highest
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

Alternative (with OFFSET in some DBs):

```
SELECT salary
FROM employees
ORDER BY salary DESC
OFFSET 1 ROW FETCH NEXT 1 ROW ONLY;
```

QA use case: Useful in validating ranking logic in leaderboards or reports.

9. How do you update multiple columns in a single query?

Answer:

```
UPDATE employees
SET salary = salary * 1.1,
    department = 'Engineering'
WHERE department = 'IT';
```

QA relevance: Sometimes needed for bulk test data modification.

10. What is normalization and why is it important in databases?

Answer:

Normalization is the process of organizing data to reduce redundancy and improve data integrity.

Common forms:

- **1NF** – No repeating groups, atomic values.
- **2NF** – No partial dependency.
- **3NF** – No transitive dependency.

QA importance: Understanding normalization helps in writing joins for related data correctly.

11. What is denormalization, and when might QA encounter it?

Answer:

Denormalization is the process of combining tables or adding redundant data to improve read performance at the expense of storage space.

Example:

- Instead of keeping customer details in a separate table and joining every time, some systems store them directly in the `orders` table for faster reporting.

QA relevance: When testing reporting systems or data warehouses, denormalization is common — you need to ensure data consistency across multiple redundant storage locations.

12. How do you get the count of rows without using `COUNT (*)`?

Answer:

You can use an identity column's `MAX()` or check row existence:

```
SELECT MAX(id) FROM employees;
```

But note — this method fails if rows have been deleted.

QA use case: In certain performance-heavy queries, testers may use alternative approaches for faster validation.

13. Explain the difference between `UNION` and `UNION ALL`.

Answer:

- **UNION** – Combines results from two queries and removes duplicates.
- **UNION ALL** – Combines results and keeps duplicates.

Example:

```
SELECT city FROM customers
UNION
SELECT city FROM suppliers;
```

QA note: If you're validating reports that should include duplicates (e.g., sales transactions), `UNION ALL` is the right choice.

14. How do you retrieve the top 5 records from a table?

Answer:

- In SQL Server:

```
SELECT TOP 5 * FROM employees ORDER BY salary DESC;
```

- In MySQL/PostgreSQL:

```
SELECT * FROM employees ORDER BY salary DESC LIMIT 5;
```

QA note: Often used in ranking verification or leaderboards.

15. How do you check for `NULL` values in SQL?

Answer:

```
SELECT * FROM employees WHERE department IS NULL;
```

QA use case: Important for testing fields where null handling is part of the business rule (e.g., optional vs mandatory fields).

16. How do you replace `NULL` values with a default value?

Answer:

```
SELECT COALESCE(department, 'Not Assigned') AS department_name
FROM employees;
```

QA note: Useful when validating how missing data is displayed in the application.

17. What is the difference between `CHAR` and `VARCHAR`?

Answer:

- **CHAR(n)** – Fixed-length storage. Always takes *n* bytes even if data is shorter.
 - **VARCHAR(n)** – Variable-length storage. Uses only the space required.
- QA relevance:** Helps understand storage behavior when validating DB schemas.
-

18. How do you retrieve the last inserted record?

Answer:

- MySQL:

```
SELECT * FROM employees ORDER BY id DESC LIMIT 1;
SELECT TOP 1 * FROM employees ORDER BY id DESC;
```

QA use case: After inserting a test record, you can confirm it was correctly added.

19. What is an index in SQL?

Answer:

An index is a database object that improves query performance by allowing faster retrieval of rows.

Types:

- **Clustered** – Changes the physical storage order.
 - **Non-clustered** – Keeps a separate lookup structure.
- QA importance:** Knowing indexes helps explain why some queries are slow during performance testing.
-

20. How do you find the total salary for each department?

Answer:

```
SELECT department, SUM(salary) AS total_salary
FROM employees
GROUP BY department;
```

QA use case: Used when validating aggregated reports.

21. How do you fetch all records where a column contains a specific substring?

Answer:

```
SELECT * FROM employees WHERE name LIKE '%John%';
```

QA note: Common when verifying search functionalities.

22. How do you find employees who don't have any orders?

Answer:

```
SELECT e.employee_id, e.name  
FROM employees e  
LEFT JOIN orders o ON e.employee_id = o.employee_id  
WHERE o.order_id IS NULL;
```

QA use case: Helpful in testing negative scenarios.

23. How do you retrieve distinct department names from a table?

Answer:

```
SELECT DISTINCT department FROM employees;
```

QA note: Prevents duplicate results when validating dropdown options.

24. What's the difference between **BETWEEN** and **IN**?

Answer:

- **BETWEEN** – For ranges (inclusive).

```
SELECT * FROM employees WHERE salary BETWEEN 5000 AND 10000;
```

- **IN** – For specific values.

```
SELECT * FROM employees WHERE department IN ('HR', 'IT');
```

QA note: Choosing the right filter ensures accurate query results.

25. How do you find the number of employees hired each year?

```
SELECT YEAR(hire_date) AS hire_year, COUNT(*) AS num_employees
FROM employees
GROUP BY YEAR(hire_date);
```

QA use case: Used when validating historical trends.

26. How do you select records with salaries higher than the department average?

Answer:

```
SELECT *
FROM employees e
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department = e.department
);
```

QA note: Useful when testing “top performer” logic.

27. What is a self-join and when is it used?

Answer:

A self-join is when a table is joined with itself to compare rows.

Example:

```
SELECT e1.name, e2.name AS manager
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.employee_id;
```

QA use case: Useful when verifying hierarchical relationships.

28. How do you get all records except the highest salary?

```
SELECT *
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

QA note: Often used in ranking validations.

29. What is a correlated subquery?

Answer:

A correlated subquery depends on the outer query for its values.

Example:

```
SELECT e1.name, e1.salary
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e1.department = e2.department
);
```

QA note: Used when comparing row-specific values to group aggregates.

30. How do you check if two tables have the same data?

Answer:

```
SELECT * FROM table1
EXCEPT
SELECT * FROM table2;

SELECT * FROM table2
EXCEPT
SELECT * FROM table1;
```

QA use case: Validating data migration or ETL processes.

31. How do you find employees whose salary is within the top 3 highest salaries in the company?

Answer:

You can use `DISTINCT` with `ORDER BY` and `LIMIT` (or `TOP` in SQL Server):

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 3;
```

Then use this in a `WHERE IN`:

```
SELECT *
FROM employees
WHERE salary IN (
    SELECT DISTINCT salary
    FROM employees
    ORDER BY salary DESC
    LIMIT 3
);
```

);

QA relevance: This is a common check when validating leaderboard rankings or incentive eligibility.

32. How do you check if a table has any rows without fetching the data?

Answer:

```
SELECT CASE WHEN EXISTS (SELECT 1 FROM employees)
            THEN 'Data exists'
            ELSE 'No data' END;
```

QA note: Helps in quick verification during automation without fetching full datasets.

33. What is the difference between **EXISTS** and **IN**?

Answer:

- **IN** – Retrieves a list of values and checks if a column matches any of them.
 - **EXISTS** – Returns TRUE if a subquery returns any rows, often more efficient for large datasets.
- Example:

```
SELECT name FROM employees
WHERE department_id IN (SELECT id FROM departments);
```

```
SELECT name FROM employees e
WHERE EXISTS (SELECT 1 FROM departments d WHERE e.department_id = d.id);
```

QA use case: Choosing between the two can improve query speed in backend validation.

34. How do you get the difference between two dates in SQL?

Answer:

- MySQL:

```
SELECT DATEDIFF(end_date, start_date) AS days_diff FROM projects;
```

- SQL Server:

```
SELECT DATEDIFF(DAY, start_date, end_date) AS days_diff FROM projects;
```

QA note: Useful when verifying SLA breaches or duration calculations.

35. How do you find the highest salary in each department?

Answer:

```
SELECT department, MAX(salary) AS max_salary
FROM employees
GROUP BY department;
```

QA use case: Validating top-earner logic per group.

36. What is a view in SQL, and why might QA use it?

Answer:

A **view** is a virtual table created from a query.

Example:

```
CREATE VIEW high_salary_employees AS
SELECT * FROM employees WHERE salary > 10000;
```

QA uses:

- To simplify complex queries during testing.
 - To create a filtered dataset for regression tests.
-

37. How do you find records present in one table but not in another?

Answer:

```
SELECT *
FROM table1
WHERE id NOT IN (SELECT id FROM table2);
```

Or with LEFT JOIN:

```
SELECT t1.*
FROM table1 t1
LEFT JOIN table2 t2 ON t1.id = t2.id
WHERE t2.id IS NULL;
```

QA note: Essential for validating data migrations.

38. How do you select random rows from a table?

Answer:

- MySQL:

```
SELECT * FROM employees ORDER BY RAND() LIMIT 5;
```

- SQL Server:

```
SELECT TOP 5 * FROM employees ORDER BY NEWID();
```

QA use case: Useful for random sampling in test data validation.

39. How do you check for duplicate rows based on multiple columns?

Answer:

```
SELECT first_name, last_name, COUNT(*)  
FROM employees  
GROUP BY first_name, last_name  
HAVING COUNT(*) > 1;
```

QA note: Prevents duplicate data entry validation failures.

40. How do you delete duplicate rows but keep one instance?

Answer:

```
DELETE e1  
FROM employees e1  
JOIN employees e2  
ON e1.email = e2.email AND e1.id > e2.id;
```

QA relevance: Common when cleaning up test databases.

41. How do you convert a string to uppercase in SQL?

Answer:

```
SELECT UPPER(name) FROM employees;
```

QA note: Useful when testing case-insensitive search.

42. How do you find the length of a string in SQL?

Answer:

- MySQL:

```
SELECT LENGTH(name) FROM employees;
```

- SQL Server:

```
SELECT LEN(name) FROM employees;
```

QA note: Used in field length validation.

43. How do you find all employees hired in the last 30 days?

Answer:

- MySQL:

```
SELECT * FROM employees
WHERE hire_date >= CURDATE() - INTERVAL 30 DAY;
```

- SQL Server:

```
SELECT * FROM employees
WHERE hire_date >= DATEADD(DAY, -30, GETDATE());
```

QA use case: Useful in validating “recently added” features.

44. How do you use a case statement in SQL?

Answer:

```
SELECT name,
       CASE
           WHEN salary > 10000 THEN 'High'
           WHEN salary BETWEEN 5000 AND 10000 THEN 'Medium'
           ELSE 'Low'
       END AS salary_range
FROM employees;
```

QA note: Helps in categorization-based validations.

45. How do you handle special characters in SQL queries?

Answer:

Escape them using backslash or double quotes depending on DB:

```
SELECT * FROM employees WHERE name LIKE '%\_%';
```

QA relevance: Needed for search feature testing.

46. How do you select only numeric values from a column containing mixed data?

Answer:

In MySQL:

```
SELECT * FROM table_name WHERE column_name REGEXP '^[0-9]+$';
```

QA note: Used when validating number-only input fields.

47. How do you join more than two tables in SQL?

Answer:

```
SELECT o.order_id, c.customer_name, p.product_name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN products p ON o.product_id = p.product_id;
```

QA note: Often used in end-to-end data verification.

48. How do you pivot data in SQL?

Answer:

```
SELECT *
FROM (SELECT department, salary FROM employees) AS src
PIVOT (AVG(salary) FOR department IN ([IT], [HR], [Finance])) AS pvt;
```

QA use case: Needed in report testing for aggregated data.

49. How do you handle transactions in SQL for testing?

Answer:

```
BEGIN TRANSACTION;
UPDATE employees SET salary = salary * 1.1;
ROLLBACK; -- to undo
```

QA note: Transactions are useful for temporary data changes during testing without permanent effects.

50. What are stored procedures and how can QA test them?

Answer:

A stored procedure is a precompiled collection of SQL statements stored in the database.

```
CREATE PROCEDURE GetHighSalaryEmployees()  
BEGIN  
    SELECT * FROM employees WHERE salary > 10000;  
END;
```

QA Testing Approach:

- Call the procedure with various inputs.
- Validate outputs match expected results.
- Check for performance issues.

Java Basics

1. What is Java, and why is it used in automation testing?

Answer:

Java is a high-level, object-oriented programming language known for its **platform independence**, meaning code written once can run anywhere with a Java Virtual Machine (JVM).

In automation testing, Java is popular because:

- It has **rich libraries** for web automation (Selenium, Rest Assured).
- It is **object-oriented**, making test scripts reusable and maintainable.
- It supports **multithreading** for parallel test execution.
- It integrates easily with **Maven**, **TestNG**, and **Jenkins** for CI/CD pipelines.

Example in Selenium:

```
System.out.println("Hello, Automation!");
```

2. What is the difference between JDK, JRE, and JVM?

Answer:

- **JDK (Java Development Kit)** – Includes JRE + development tools (compiler `javac`) for creating Java programs.
- **JRE (Java Runtime Environment)** – Provides JVM + libraries to run Java programs (no compiler).
- **JVM (Java Virtual Machine)** – Converts Java bytecode into machine code for execution.

In QA:

- You install the **JDK** to write and compile automation scripts.
 - Your code runs in the **JVM** during test execution.
-

3. What are Java Data Types, and how are they used in test automation?

Answer:

Java has **primitive** and **non-primitive** data types.

Primitive: byte, short, int, long, float, double, char, boolean

Non-primitive: String, Arrays, Classes, Interfaces

Example:

```
int testCount = 5; // primitive
String testName = "Login Test"; // non-primitive
boolean status = true;
```

In QA, data types store:

- Test counts
 - Test case names
 - Pass/Fail flags
-

4. What is the difference between primitive and reference data types?

Answer:

- **Primitive types** store actual values directly in memory (e.g., `int x = 10;` stores 10).
 - **Reference types** store memory addresses pointing to objects (e.g., `String s = new String("Test");` stores a reference to the string object in heap memory).
-

5. What are Java Operators?

Answer:

Java operators are special symbols that perform operations on variables/values.

- **Arithmetic:** +, -, *, /, %
- **Relational:** ==, !=, >, <, >=, <=
- **Logical:** &&, ||, !
- **Assignment:** =, +=, -=
- **Increment/Decrement:** ++, --

Example in Selenium:

java

```
CopyEdit
if (driver.getTitle().equals("Dashboard") && status == true) {
    System.out.println("Test Passed");
}
```

6. Explain Conditional Statements in Java.

Answer:

Conditional statements control the flow of execution based on conditions:

```
if (condition) { }
else if (condition) { }
else { }
```

Example:

```
if (statusCode == 200) {
    System.out.println("API Test Passed");
} else {
    System.out.println("API Test Failed");
}
```

7. What is the difference between if-else and switch-case?

Answer:

- **if-else:** Best for range-based or complex conditions.
- **switch-case:** Best for fixed value checks.

Example:

```
switch(statusCode) {
    case 200: System.out.println("OK"); break;
    case 404: System.out.println("Not Found"); break;
    default: System.out.println("Unknown Status");
}
```

8. What are Loops in Java, and how are they used in QA?

Answer:

Loops repeat actions until a condition is met.

- **for loop** – Known iterations
- **while loop** – Runs while condition is true
- **do-while loop** – Runs at least once

Example:

```
for (int i = 0; i < testCases.size(); i++) {
    runTest(testCases.get(i));
}
```

9. Difference between break and continue in loops?

Answer:

- **break** – Stops the loop completely.
- **continue** – Skips the current iteration and moves to the next.

Example:

```
for (String tc : testCases) {  
    if (tc.contains("Deprecated")) continue;  
    executeTest(tc);  
}
```

10. What is an Array in Java?

Answer:

An array is a fixed-size collection of similar data types.

Example:

```
String[] browsers = {"Chrome", "Firefox", "Edge"};  
for (String browser : browsers) {  
    System.out.println(browser);  
}
```

In QA, arrays are used to store:

- Multiple test inputs
- Browser names
- API endpoints

11. Difference between Array and ArrayList in Java?

Answer:

- **Array:** Fixed size, can store both primitive and object types.
- **ArrayList:** Dynamic size, can store only objects (wrapper classes for primitives).

Example in QA:

```
// Array  
String[] browsers = {"Chrome", "Firefox"};  
  
// ArrayList  
ArrayList<String> testCases = new ArrayList<>();  
testCases.add("Login Test");  
testCases.add("Checkout Test");
```

Automation projects prefer **ArrayList** because test case counts often change.

12. What is a String in Java, and why is it immutable?

Answer:

- A **String** is a sequence of characters stored as an object.
- **Immutable** means once created, the value cannot be changed.
- If you modify it, a new object is created in memory.

Example:

```
String test = "Login";  
test = test + " Test"; // New object created
```

In QA, immutability ensures test data consistency during execution.

13. Difference between String, StringBuilder, and StringBuffer?

Answer:

- **String**: Immutable, slower in frequent modifications.
- **StringBuilder**: Mutable, faster, not thread-safe.
- **StringBuffer**: Mutable, thread-safe but slower than StringBuilder.

Example:

```
StringBuilder sb = new StringBuilder("Test");  
sb.append(" Passed");
```

In automation, **StringBuilder** is preferred for building large API requests dynamically.

14. What is Object-Oriented Programming (OOP) in Java?

Answer:

OOP is a programming paradigm that organizes code into objects containing:

1. **Encapsulation** – Wrapping data & methods.
2. **Inheritance** – Reusing code from parent classes.
3. **Polymorphism** – Methods behaving differently based on object.
4. **Abstraction** – Hiding implementation details.

QA Benefit: Makes automation frameworks modular & maintainable.

15. Explain Encapsulation with an example.

Answer:

Encapsulation means **hiding data** using `private` variables and **exposing access** via getters/setters.

```
public class TestCase {
    private String name;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

In QA, encapsulation is used for storing sensitive test data (like credentials).

16. Explain Inheritance with an example.

Answer:

Inheritance allows one class to use properties/methods of another.

```
class BaseTest {
    public void setup() { System.out.println("Launching Browser"); }
}
class LoginTest extends BaseTest {
    public void runTest() { System.out.println("Executing Login Test"); }
}
```

QA frameworks use **BaseTest** classes for common setup/teardown methods.

17. Explain Polymorphism with examples.

Answer:

Two types:

- **Compile-time (Overloading)** – Same method name, different parameters.
- **Runtime (Overriding)** – Same method signature, different implementation.

Example:

```
class TestRunner {
    void runTest(String name) { System.out.println(name); }
    void runTest(String name, int id) { System.out.println(name + " " + id); }
}
```

Used in QA when writing **generic test methods** for different test cases.

18. Explain Abstraction with an example.

Answer:

Hiding internal details, showing only essential methods.

```
abstract class Browser {
    abstract void launch();
}
class Chrome extends Browser {
    void launch() { System.out.println("Launching Chrome"); }
}
```

QA uses abstraction in framework **interfaces and abstract classes**.

19. Difference between Abstract Class and Interface?

Answer:

Feature	Abstract Class	Interface
Methods	Can have abstract & concrete methods	Only abstract (Java 7) / default + static (Java 8+)
Variables	Can be final/non-final	Always <code>public static final</code>
Inheritance	Single	Multiple

Automation frameworks often use **interfaces** for WebDriver event handling.

20. What is Method Overloading and Overriding?

Answer:

- **Overloading:** Same method name, different parameters (compile-time).
 - **Overriding:** Same method name/signature in subclass (runtime).
-

21. What is the difference between final, finally, and finalize()?

Answer:

- **final:** Keyword for constant variables or preventing inheritance.
 - **finally:** Code block that always executes after try-catch.
 - **finalize():** Method called by Garbage Collector before object destruction.
-

22. What is Exception Handling in Java?

Answer:

Mechanism to handle runtime errors without crashing the program.

```
try {
    int num = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero");
} finally {
    System.out.println("Cleanup code");
}
```

In QA, exception handling prevents script failure on minor issues.

23. Difference between Checked and Unchecked Exceptions?

Answer:

- **Checked:** Checked at compile time (IOException, SQLException).
 - **Unchecked:** Checked at runtime (NullPointerException, ArithmeticException).
-

24. What is a Wrapper Class in Java?

Answer:

Object representation of primitive types.

Example:

```
int x = 5;
Integer y = Integer.valueOf(x);
```

Used in Collections, as they store only objects.

25. What is Autoboxing and Unboxing?

Answer:

- **Autoboxing:** Converting primitive to wrapper automatically.
 - **Unboxing:** Converting wrapper to primitive automatically.
-

26. What are Collections in Java?

Answer:

A framework for storing/managing groups of objects.

- **List** – Ordered, allows duplicates.
 - **Set** – Unordered, no duplicates.
 - **Map** – Key-value pairs.
-

27. Difference between List and Set?

Answer:

- **List:** Allows duplicates, maintains insertion order.
 - **Set:** No duplicates, no guaranteed order.
-

28. Difference between HashMap and HashSet?

Answer:

- **HashMap:** Stores key-value pairs.
 - **HashSet:** Stores only unique values.
-

29. How to iterate over a Collection?

Answer:

```
for (String test : testCases) {  
    System.out.println(test);  
}
```

Or using **Iterator**:

```
Iterator<String> it = testCases.iterator();  
while (it.hasNext()) {  
    System.out.println(it.next());  
}
```

30. Difference between ArrayList and LinkedList?

Answer:

- **ArrayList:** Better for searching.
 - **LinkedList:** Better for insertion/deletion.
-

31. What is a HashMap in Java?

Answer:

Stores data in key-value format.

```
HashMap<Integer, String> map = new HashMap<>();  
map.put(1, "Login Test");
```

QA uses HashMap for storing test case data dynamically.

32. What is Synchronization in Java?

Answer:

Process of controlling access to shared resources in multi-threading.

33. How is multi-threading used in automation?

Answer:

Runs multiple tests in parallel to save execution time (TestNG `parallel="methods"`).

34. What is the difference between throw and throws?

Answer:

- **throw:** Used to explicitly throw an exception.
 - **throws:** Declares exceptions in method signature.
-

35. What is the difference between == and .equals()?

Answer:

- **==:** Compares memory references.
 - **.equals():** Compares object values.
-

36. What are Static Methods and Variables?

Answer:

Belong to the class, not instances. Used for utility methods in frameworks.

37. What are Constructors in Java?

Answer:

Special methods used to initialize objects.

38. Difference between default and parameterized constructors?

Answer:

- **Default:** No parameters.
 - **Parameterized:** Accepts arguments to set values.
-

39. What is the 'this' keyword in Java?

Answer:

Refers to the current object instance.

40. What is the 'super' keyword in Java?

Answer:

Refers to the parent class object/method.

41. What is File Handling in Java?

Answer:

Reading/writing files using `File`, `FileReader`, `BufferedReader`, etc.

In QA, used for reading test data from `.txt` or `.csv`.

42. How to read a file in Java?

Answer:

```
BufferedReader br = new BufferedReader(new FileReader("data.txt"));
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
br.close();
```

43. How to write into a file in Java?

Answer:

```
FileWriter writer = new FileWriter("output.txt");  
writer.write("Test Passed");  
writer.close();
```

44. What is Serialization in Java?

Answer:

Converting object into byte stream for storage/transfer.

45. What is Deserialization in Java?

Answer:

Converting byte stream back to object.

46. What are Java Streams?

Answer:

Used for processing collections in a functional style.

```
list.stream().filter(x -> x.startsWith("T")).forEach(System.out::println);
```

47. Difference between Lambda Expressions and Anonymous Class?

Answer:

- **Lambda:** Short syntax for functional interfaces.
 - **Anonymous class:** Full class declaration without name.
-

48. How to handle dates in Java?

Answer:

Using `LocalDate`, `LocalDateTime`, `SimpleDateFormat`.

```
LocalDate today = LocalDate.now();
```

49. How to generate random data in Java?

Answer:

```
Random rand = new Random();  
int num = rand.nextInt(100);
```

Useful for test data generation.

50. How is Java integrated in Selenium Automation?

Answer:

- Java is the **language** to write test scripts.
- Selenium is the **library** that interacts with the browser.

Example:

```
WebDriver driver = new ChromeDriver();  
driver.get("https://example.com");
```

Basic Java Coding Examples

1. Reverse a String

```
public class ReverseString {
    public static void main(String[] args) {
        String str = "Automation";
        String rev = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            rev += str.charAt(i);
        }
        System.out.println("Reversed: " + rev);
    }
}
```

Why QA needs this: Useful when validating reverse data transformations in APIs or UI.

2. Check if a String is a Palindrome

```
public class PalindromeCheck {
    public static void main(String[] args) {
        String str = "madam";
        String rev = new StringBuilder(str).reverse().toString();
        if (str.equals(rev))
            System.out.println("Palindrome");
        else
            System.out.println("Not Palindrome");
    }
}
```

QA use: Checking same input/output values in transformations.

3. Count Vowels in a String

```
public class CountVowels {
    public static void main(String[] args) {
        String str = "Quality Assurance";
        int count = 0;
        for (char c : str.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(c) != -1) count++;
        }
        System.out.println("Vowels: " + count);
    }
}
```

QA use: Validating text data.

4. Swap Two Numbers Without Temporary Variable

```
public class SwapNumbers {
    public static void main(String[] args) {
```

```

        int a = 5, b = 10;
        a = a + b;
        b = a - b;
        a = a - b;
        System.out.println("a=" + a + ", b=" + b);
    }
}

```

QA use: Testing logic manipulation without extra memory.

5. Find Factorial of a Number

```

public class Factorial {
    public static void main(String[] args) {
        int num = 5, fact = 1;
        for (int i = 1; i <= num; i++) {
            fact *= i;
        }
        System.out.println("Factorial: " + fact);
    }
}

```

QA use: Mathematical validations in reports or APIs.

6. Fibonacci Series

```

public class Fibonacci {
    public static void main(String[] args) {
        int n = 7, a = 0, b = 1;
        System.out.print(a + " " + b);
        for (int i = 2; i < n; i++) {
            int c = a + b;
            System.out.print(" " + c);
            a = b; b = c;
        }
    }
}

```

QA use: Data pattern validation.

7. Find Largest Number in Array

```

public class LargestInArray {
    public static void main(String[] args) {
        int[] arr = {10, 45, 23, 78, 56};
        int max = arr[0];
        for (int num : arr) {
            if (num > max) max = num;
        }
        System.out.println("Largest: " + max);
    }
}

```

QA use: Finding max values in datasets.

8. Reverse an Array

```
public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

QA use: Data order validation.

9. Count Words in a String

```
public class WordCount {
    public static void main(String[] args) {
        String str = "Automation testing with Java";
        String[] words = str.trim().split("\\s+");
        System.out.println("Words: " + words.length);
    }
}
```

QA use: Checking text fields.

10. Remove Duplicates from Array

```
import java.util.HashSet;
public class RemoveDuplicates {
    public static void main(String[] args) {
        int[] arr = {1, 2, 2, 3, 4, 4};
        HashSet<Integer> set = new HashSet<>();
        for (int n : arr) set.add(n);
        System.out.println(set);
    }
}
```

QA use: Ensuring unique records.

11. Find Duplicate Elements in Array

```
public class FindDuplicates {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 4, 5, 1};
        for (int i = 0; i < arr.length; i++) {
            for (int j = i+1; j < arr.length; j++) {
                if (arr[i] == arr[j])
                    System.out.println("Duplicate: " + arr[i]);
            }
        }
    }
}
```

12. Check Prime Number

```
public class PrimeCheck {
    public static void main(String[] args) {
        int num = 13;
        boolean prime = true;
        for (int i = 2; i <= num/2; i++) {
            if (num % i == 0) { prime = false; break; }
        }
        System.out.println(prime ? "Prime" : "Not Prime");
    }
}
```

13. Reverse Words in a Sentence

```
public class ReverseWords {
    public static void main(String[] args) {
        String str = "I love Java";
        String[] words = str.split(" ");
        for (int i = words.length - 1; i >= 0; i--) {
            System.out.print(words[i] + " ");
        }
    }
}
```

14. Check Anagram Strings

```
import java.util.Arrays;
public class AnagramCheck {
    public static void main(String[] args) {
        String s1 = "listen", s2 = "silent";
        char[] a1 = s1.toCharArray();
        char[] a2 = s2.toCharArray();
        Arrays.sort(a1);
        Arrays.sort(a2);
        System.out.println(Arrays.equals(a1, a2) ? "Anagram" : "Not Anagram");
    }
}
```

15. Find Second Largest Number

```
public class SecondLargest {
    public static void main(String[] args) {
        int[] arr = {10, 45, 78, 56};
        int largest = Integer.MIN_VALUE, second = Integer.MIN_VALUE;
        for (int num : arr) {
            if (num > largest) {
                second = largest;
                largest = num;
            } else if (num > second && num != largest) {
                second = num;
            }
        }
        System.out.println("Second Largest: " + second);
    }
}
```

16. Sum of Digits

```
public class SumDigits {
    public static void main(String[] args) {
        int num = 12345, sum = 0;
        while (num > 0) {
            sum += num % 10;
            num /= 10;
        }
        System.out.println("Sum: " + sum);
    }
}
```

17. Remove White Spaces from String

```
public class RemoveSpaces {
    public static void main(String[] args) {
        String str = " Hello World ";
        System.out.println(str.replaceAll("\\s+", ""));
    }
}
```

18. Convert String to Integer

```
public class StringToInt {
    public static void main(String[] args) {
        String s = "123";
        int num = Integer.parseInt(s);
        System.out.println(num + 10);
    }
}
```

19. Find Missing Number in Array

```
public class MissingNumber {
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5};
        int n = 5, total = n*(n+1)/2, sum = 0;
        for (int num : arr) sum += num;
        System.out.println("Missing: " + (total - sum));
    }
}
```

20. Read File and Print Content

```
import java.io.*;
public class ReadFile {
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new FileReader("data.txt"));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        br.close();
    }
}
```

QA use: Reading test data from a file.

Git & GitHub

1. What is Git and why is it important for QA engineers?

Answer:

Git is a **distributed version control system** used to track changes in source code over time. It allows multiple developers and testers to work on the same project without overwriting each other's work.

Why it's important for QA:

- QA can manage their automation test scripts effectively.
- Enables rollback to a stable test version if new code breaks tests.
- Supports branching, allowing QA to test features in isolation.
- Works seamlessly with CI/CD tools like Jenkins for automated test execution.

Example:

If a Selenium automation test breaks after a developer pushes new changes, QA can use:

```
git log
git checkout <commit-id>
```

to roll back and confirm if the issue is due to the latest code.

2. Difference between Git and GitHub.

Answer:

Feature	Git	GitHub
Definition	Local version control tool	Cloud hosting service for Git repositories
Location	Installed locally	Web-based
Function	Tracks and manages code changes	Collaboration, Pull Requests, Issues
Offline Support	Yes	No

QA Usage:

- Use Git to commit automation test updates locally.
 - Push code to GitHub for team review and CI/CD pipeline execution.
-

3. Explain the Git branching model for QA.

Answer:

A common model is **Git Flow**:

- **main/master**: Stable production code.
- **develop**: Integration branch.
- **feature/**: New feature branches.
- **release/**: Pre-production testing branches.
- **hotfix/**: Urgent bug fixes.

QA Workflow Example:

1. Developer creates `feature/user-login`.
2. QA checks out that branch and runs automation tests.
3. If all tests pass, QA approves merging into `develop`.

4. How do you resolve merge conflicts in Git?

Answer:

Steps for QA:

1. Attempt merge:

```
git merge develop
```

2. Git marks conflict lines in the file:

```
<<<<<<< HEAD
old test code
=====
updated test code
>>>>>>> feature-branch
```

3. Manually choose correct code.

4. Save and run:

```
git add .
git commit -m "Resolved merge conflict in loginTest.java"
git push
```

5. How do you revert a commit without losing history?

Answer:

Use `git revert <commit-id>` instead of `git reset` to keep commit history.

QA Example:

Automation script change introduced flaky tests:

```
git log
git revert alb2c3d
```

6. Difference between `git merge` and `git rebase`.

Answer:

- **Merge:** Combines two branches and keeps commit history intact.
- **Rebase:** Re-applies commits onto another branch, creating a linear history.

QA Tip:

Merge is safer for shared automation branches since it preserves change history.

7. How to clone a repository from GitHub?

Answer:

```
git clone https://github.com/user/repo.git
```

QA Use:

Clone automation framework repository before starting regression tests.

8. What is a pull request (PR) and why QA uses it?

Answer:

PR is a request to merge code changes from one branch to another.

QA Benefits:

- Review automation script changes.
 - Trigger test runs before merge.
 - Discuss potential issues with developers.
-

9. How to stash changes in Git?

Answer:

Command:

```
git stash
git pull
git stash pop
```

QA Use:

You're writing a test case but need to pull urgent changes from `develop` without losing work.

10. Explain Git tags and QA usage.

Answer:

Tags mark specific commits, usually for releases.

Example:

```
git tag v1.0.0
git push origin v1.0.0
```

QA can tag stable automation builds.

11. What is the difference between `git fetch` and `git pull`?

Answer:

- **git fetch:** Downloads latest changes but doesn't merge them.
- **git pull:** Fetches and merges changes automatically.

QA Tip:

Use `git fetch` to review changes before merging to avoid breaking tests.

12. How do you view commit history in Git?

Answer:

```
git log --oneline --graph --all
```

QA can check when a specific test case was last modified.

13. How do you create and switch to a new branch in Git?

Answer:

```
git checkout -b new-automation-tests
```

QA uses branches to create isolated test cases for a feature.

14. How do you delete a branch in Git?

Answer:

```
git branch -d old-branch    # local
git push origin --delete old-branch    # remote
```

QA deletes obsolete automation branches to keep repo clean.

15. How do you see differences between two commits?

Answer:

```
git diff commit1 commit2
```

QA can compare changes in a test file to identify why a test is failing.

16. How do you undo the last commit but keep changes?

Answer:

```
git reset --soft HEAD~1
```

Useful for QA when commit message was incorrect.

17. What is `.gitignore` and why QA uses it?

Answer:

`.gitignore` lists files Git should ignore.

QA Example:

Ignore `test-output/` reports so they don't clutter the repo.

18. How do you check which branch you are on?

Answer:

```
git branch
```

QA uses this to ensure they're testing the correct branch.

19. How to rename a branch in Git?

Answer:

```
git branch -m old-name new-name
```

QA renames branches for clarity (e.g., `feature-tests` → `login-tests`).

20. How do you see who made changes to a file?

Answer:

```
git blame filename
```

QA can find the developer who last modified a failing test case.

CI/CD pipelines

21. What is CI/CD and why is it important for QA engineers?

Answer:

CI/CD stands for **Continuous Integration** and **Continuous Delivery/Deployment**.

- **Continuous Integration (CI):** Developers frequently merge their code changes into a shared repository. Each merge triggers automated builds and tests.
- **Continuous Delivery (CD):** After passing tests, code is packaged and made ready for deployment. Deployment is triggered manually.
- **Continuous Deployment:** Every change that passes tests is deployed automatically without manual approval.

Importance for QA:

- Automation scripts are run immediately after code changes, catching defects early.
- Builds are consistent and tested in the same environment every time.
- QA can spend less time on repetitive regression tests and more on exploratory testing.

QA Example:

A Jenkins CI pipeline runs Selenium and API tests every time a developer commits code. If a bug is detected, QA gets instant feedback and can raise a defect without waiting for manual builds.

22. What is the difference between Continuous Delivery and Continuous Deployment?

Answer:

Feature	Continuous Delivery	Continuous Deployment
Deployment	Manual trigger after QA approval	Fully automated to production

Feature	Continuous Delivery	Continuous Deployment
Risk Level	Lower	Higher
QA Involvement	Approves and validates releases	Monitors production after release
Example	QA tests in staging before pushing to live	Code goes live right after automated tests pass

QA Perspective:

Continuous Delivery gives QA more control over when changes go live, while Continuous Deployment relies entirely on automation and monitoring.

23. What is the role of QA in a CI/CD pipeline?

Answer:

QA's responsibilities in CI/CD include:

1. **Integrating automated tests** (UI, API, performance) into the pipeline.
2. **Defining quality gates** so builds fail if tests fail.
3. **Monitoring pipeline runs** to detect flaky or broken tests.
4. **Collaborating with DevOps** to set up staging/test environments.
5. **Reviewing test reports** to ensure full coverage before deployment.

Example:

In a Jenkins pipeline, QA ensures that Postman API tests run before Selenium UI tests to catch backend issues early.

24. How does CI/CD improve software quality for QA teams?

Answer:

- **Faster feedback loop** → Bugs found earlier.
- **Consistent environments** → Less “works on my machine” issues.
- **Reduced manual effort** → Regression tests automated.
- **Release confidence** → Only tested code is deployed.

Example:

A failing login test in CI alerts QA before release, preventing a production outage.

25. What are quality gates in CI/CD and how does QA use them?

Answer:

Quality gates are **conditions that must be met before code can be deployed**.

Examples for QA:

- All unit & integration tests must pass.
- Automation pass rate $\geq 95\%$.
- Code coverage $\geq 80\%$.
- No critical severity bugs in the issue tracker.

In Jenkins, QA can configure pipelines to fail automatically if these conditions aren't met.

26. How do you integrate automated tests in a CI/CD pipeline?

Answer:

Steps for QA:

1. Push automation scripts to a Git repo.
2. Configure Jenkins or GitHub Actions to pull code on every commit.
3. Install dependencies (Java, Maven, Node.js, etc.).
4. Run tests:

```
bash
CopyEdit
mvn clean test
```

5. Generate test reports (Allure, ExtentReports).
 6. Set build to fail if tests fail.
-

27. How do you trigger a CI/CD pipeline automatically after code changes?

Answer:

Use **webhooks** or **polling**.

- In GitHub → Jenkins, enable “Build when a change is pushed to GitHub”.
- Alternatively, Jenkins can poll the repo at intervals.

QA Example:

Whenever an automation script is updated in GitHub, Jenkins triggers regression testing automatically.

28. What are the common stages of a CI/CD pipeline from a QA perspective?

Answer:

1. **Build** → Compile code.
2. **Unit Tests** → Quick checks.
3. **Integration/API Tests** → QA validates backend logic.
4. **UI Tests** → Selenium/Playwright tests.

5. **Performance Tests** → JMeter/Gatling.
6. **Deploy to Staging** → QA performs final checks.
7. **Deploy to Production** → Final release.

29. What is a blue-green deployment and how does it help QA?

Answer:

Blue-green deployment means having two identical environments:

- **Blue:** Current production.
- **Green:** New version being tested.

QA tests on **green** while blue is live. If green passes, traffic switches instantly.

Benefit: No downtime and quick rollback if tests fail.

30. How do you handle flaky tests in CI/CD pipelines?

Answer:

- Identify unstable tests by reviewing multiple failed builds.
- Use retry mechanisms in test scripts.
- Improve synchronization (explicit waits in Selenium).
- Remove unnecessary dependencies on environment.

QA Example:

A Selenium test that fails randomly due to slow page load can be stabilized with:

```
java
CopyEdit
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("login")));
```

31. What tools can be used for CI/CD in QA automation?

Answer:

- **Jenkins** – Most popular, highly customizable.
- **GitHub Actions** – Easy GitHub integration.
- **GitLab CI/CD** – Built-in for GitLab projects.
- **CircleCI** – Cloud-based, fast.
- **Azure DevOps** – Great for enterprise QA pipelines.

32. How do you run parallel tests in CI/CD?

Answer:

- Use Selenium Grid or cloud services like BrowserStack/SauceLabs.
- Configure pipeline to split test cases into multiple jobs.

QA Benefit: Reduces test execution time significantly.

33. How do you generate and share test reports in CI/CD?

Answer:

- Use **Allure Reports**, **Extent Reports**, or **JUnit HTML reports**.
 - Configure Jenkins “Publish HTML Reports” post-build step.
 - Share links to reports in Slack/Teams automatically.
-

34. How do you manage test data in CI/CD?

Answer:

- Store test data in CSV, JSON, or databases.
 - Use environment variables for sensitive data.
 - Reset test data before each run to avoid pollution.
-

35. What is the difference between CI/CD for backend APIs and frontend UI testing?

Answer:

- **Backend CI/CD:** Focuses on API testing, database migrations, and integration.
- **Frontend CI/CD:** Focuses on UI testing across browsers and devices.

QA Perspective: API tests run earlier in pipeline because they are faster and catch backend issues before UI tests.

Jenkins

36. What is Jenkins and why is it important for QA engineers?

Answer:

Jenkins is an **open-source automation server** used to build, test, and deploy applications.

Importance for QA:

- Automates test execution (UI, API, performance tests).
- Integrates with GitHub/GitLab to run tests on every commit.

- Generates and publishes test reports automatically.
- Supports scheduling (nightly regression runs).
- Works with Docker, Selenium Grid, Postman, and other QA tools.

Example:

When a developer commits code, Jenkins pulls the latest changes, runs Selenium regression tests, and sends results to QA via email/Slack.

37. How do you create a basic Jenkins job to run automation tests?

Answer:

Steps:

1. Open Jenkins → “New Item”.
2. Select **Freestyle Project** → Name it (e.g., `Selenium_UI_Tests`).
3. Configure Git repository URL.
4. Set **Build Triggers**: “Build when a change is pushed to GitHub”.
5. Add **Build Steps**:

```
bash
CopyEdit
mvn clean test
```

6. Post-build actions: Publish HTML reports (Allure/Extent).

QA Example:

Run `mvn clean test` to execute Selenium + TestNG tests for UI verification.

38. What is the difference between a Freestyle job and a Pipeline job in Jenkins?

Answer:

Feature	Freestyle	Pipeline
Setup	GUI-based	Code-based (<code>Jenkinsfile</code>)
Flexibility	Limited	Highly customizable
Version Control	Cannot be versioned	Can store <code>Jenkinsfile</code> in Git
QA Usage	Simple one-step automation tests	Complex multi-stage CI/CD pipelines

QA Tip:

For simple automation, Freestyle works fine; for multi-environment testing, use Pipelines.

39. How do you configure Jenkins to trigger builds automatically from GitHub?

Answer:

1. Install **GitHub Integration Plugin** in Jenkins.
2. In GitHub repo → Settings → Webhooks → Add webhook with Jenkins URL:

```
arduino
CopyEdit
http://jenkins-server/github-webhook/
```

3. In Jenkins job → Build Triggers → Check “GitHub hook trigger for GITScm polling”.
4. Now, whenever code is pushed, Jenkins runs automatically.

QA Example:

Whenever a new Selenium test is committed, Jenkins immediately runs regression suite.

40. How do you integrate Selenium tests with Jenkins?

Answer:

1. Push Selenium + TestNG/Maven project to GitHub.
2. Configure Jenkins job to pull the repo.
3. Add build command:

```
bash
CopyEdit
mvn clean test
```

4. Configure post-build action to publish reports from `target/surefire-reports` or Allure results.
-

41. How do you parameterize a Jenkins build for QA testing?

Answer:

- Go to job config → “This project is parameterized”.
- Add parameters:
 - **Choice Parameter:** Environment (DEV/STAGE/PROD).
 - **String Parameter:** Test suite name.

QA Example:

Select `STAGE` from the dropdown to run automation on staging environment without changing code.

42. What is a Jenkinsfile and why QA uses it?

Answer:

A **Jenkinsfile** defines a pipeline as code.

Advantages for QA:

- Version-controlled with test scripts.
- Easy to replicate and modify pipelines.
- Supports multiple stages (build → test → deploy).

Example Jenkinsfile for QA:

```
groovy
CopyEdit
pipeline {
    agent any
    stages {
        stage('Test') {
            steps {
                sh 'mvn clean test'
            }
        }
        stage('Report') {
            steps {
                publishHTML([reportDir: 'target/surefire-reports', reportFiles:
'index.html'])
            }
        }
    }
}
```

43. How do you run tests in parallel in Jenkins?

Answer:

- Use **Declarative Pipeline** with `parallel` block.
- Example:

```
groovy
CopyEdit
parallel {
    stage('Chrome Tests') {
        steps { sh 'mvn test -Dbrowser=chrome' }
    }
    stage('Firefox Tests') {
        steps { sh 'mvn test -Dbrowser=firefox' }
    }
}
```

QA Benefit: Reduces execution time for cross-browser testing.

44. How do you integrate Postman API tests in Jenkins?

Answer:

1. Export Postman collection + environment JSON files.
2. Install **Newman** on Jenkins server:

```
bash
CopyEdit
npm install -g newman
```

3. Add Jenkins build step:

```
bash
CopyEdit
newman run collection.json -e environment.json --reporters cli,html
```

45. How do you schedule a nightly regression run in Jenkins?

Answer:

- In job config → Build Triggers → “Build periodically”.
- Example schedule:

```
sql
CopyEdit
H 2 * * * # Runs at 2 AM every day
```

QA Use: Automate nightly regression testing for fresh results in the morning.

46. How do you publish HTML test reports in Jenkins?

Answer:

1. Install **HTML Publisher Plugin**.
2. Post-build → Publish HTML reports.
3. Set report directory and file name.

QA Example:

Publish Allure report from `target/allure-report` after test execution.

47. How do you handle failed builds in Jenkins for QA tests?

Answer:

- Enable **Post-build actions** to send failure notifications via email/Slack.
- Use **Post-build scripts** to archive screenshots/logs.

- Set **Build status to unstable** if non-critical tests fail.
-

48. How do you use Jenkins to test across multiple environments?

Answer:

- Use parameters for environment URLs.
- Pass environment variable to test framework:

```
bash
CopyEdit
mvn test -Denvironment=stage
```

QA Example:

Run the same suite in DEV, STAGE, and PROD without code changes.

49. How do you secure Jenkins for QA projects?

Answer:

- Enable role-based access control.
 - Restrict build execution rights.
 - Mask credentials in logs.
 - Use credentials plugin for API keys and passwords.
-

50. How do you archive test artifacts in Jenkins?

Answer:

- Post-build → Archive the artifacts.
- Example: Archive `screenshots/` or `logs/` directories for failed tests.

QA Example:

Every failed Selenium test screenshot is saved and downloadable from Jenkins.

JIRA

1. What is JIRA and why do QA engineers use it?

Answer:

JIRA is a project management and issue tracking tool developed by Atlassian.

While it's widely used for Agile software development, QA engineers use it mainly for **bug tracking**, **test management**, and **project collaboration**.

For QA, JIRA is important because:

- It allows **logging defects** found during testing.
- You can **track the lifecycle of a bug** from creation to closure.
- Integrates with **test management plugins** like Zephyr, Xray, and TestRail.
- Supports **Agile boards** (Scrum, Kanban) so QA can track testing progress in sprints.
- Enables collaboration between QA, developers, and product managers.

Example: If you find a bug in the checkout process during regression testing, you can log it in JIRA with steps to reproduce, attach screenshots, assign it to a developer, and track its fix.

2. What is an Issue in JIRA?

Answer:

In JIRA, an **Issue** is the basic unit of work. It could be:

- A **bug** found in testing.
- A **story** describing a new feature.
- A **task** for a developer or QA.
- An **epic** representing a large set of related tasks.

For QA, the most common issue type is **Bug**.

Example: In a mobile app, if the "Login" button is not clickable, you would create a **Bug** issue in JIRA with all relevant details.

3. What is a Workflow in JIRA?

Answer:

A **Workflow** in JIRA defines the **lifecycle** of an issue — the sequence of steps it goes through from creation to closure.

Example of a QA bug workflow:

1. **Open/New** – Bug is reported by QA.
2. **In Progress** – Developer is fixing it.
3. **Resolved/Fixed** – Developer marks it as fixed.
4. **Ready for Testing** – QA tests the fix.
5. **Closed** – QA verifies and closes the bug.
6. **Reopened** – QA finds the bug still exists.

A good workflow helps QA track the exact status of defects.

4. How do you create a Bug in JIRA?

Answer:

Steps for QA:

1. Click **Create** on the JIRA dashboard.
2. Select **Project** and **Issue Type** as **Bug**.
3. Fill **Summary** (short title) and **Description** (detailed explanation).
4. Add **Steps to Reproduce** so the developer can replicate the bug.
5. Attach **screenshots or logs**.
6. Select **Priority** (High, Medium, Low) based on impact.
7. Assign it to the relevant developer or leave it unassigned.
8. Click **Create**.

Tip for QA: Always write clear, reproducible bug reports so developers can fix them faster.

5. What is the difference between Priority and Severity in JIRA?

Answer:

- **Priority** (Business perspective): Decides how quickly a bug should be fixed.
 - Example: A spelling mistake on the homepage might be low severity but **high priority** if a client demo is tomorrow.
- **Severity** (Technical perspective): Indicates the impact of the bug on functionality.
 - Example: A crash in the payment gateway is **high severity** but may be low priority if the feature is not released yet.

For QA, understanding both helps in correctly categorizing defects.

6. How do QA engineers use JIRA for Agile testing?

Answer:

In Agile projects:

- QA uses **Scrum boards** to track testing tasks in sprints.

- **Stories** are created for features, and **sub-tasks** for test cases.
- Bugs are linked to stories so QA can see which feature they belong to.
- QA participates in **daily standups** and updates task statuses.
- **Burndown charts** in JIRA help monitor testing progress.

Example: In Sprint 5, a login feature story might have linked sub-tasks for test case creation, execution, and bug logging.

7. Can JIRA be used for Test Case Management?

Answer:

Yes, but JIRA by itself does not have built-in test case management. QA teams use **plugins** like:

- **Zephyr for JIRA** – Allows test case creation, execution, and reporting.
- **Xray** – Manages manual and automated test cases.
- **TestRail integration** – Connects external test management with JIRA.

Example: Using Zephyr, QA can write test cases inside JIRA, execute them, and log bugs directly linked to failed tests.

8. How do you link issues in JIRA?

Answer:

QA often needs to link bugs to related stories or other bugs for traceability.

Steps:

1. Open the issue.
2. Click **More** → **Link**.
3. Choose **link type** (e.g., "relates to," "is blocked by," "duplicates").
4. Enter the issue key of the related item.

Example: If Bug #123 is caused by incomplete work in Story #456, QA links it with “is blocked by.”

9. What are JIRA Filters and how can QA use them?

Answer:

A **Filter** is a saved search query in JIRA. QA can use filters to:

- View only open bugs.
- See high-priority issues assigned to them.
- List all bugs in a sprint.

Example JQL (JIRA Query Language) for QA:

```
jql
CopyEdit
project = "E-Commerce" AND issuetype = Bug AND status != Closed ORDER BY priority DESC
```

This shows all open bugs in the "E-Commerce" project sorted by priority.

10. How do you generate reports in JIRA?

Answer:

QA can use JIRA reports to track testing and defect trends. Examples:

- **Created vs Resolved Issues** – See if bug resolution is keeping up.
- **Sprint Report** – Track completed and pending testing tasks.
- **Pie Chart Report** – Show bugs by priority or status.

Example: At the end of a sprint, QA might generate a "Created vs Resolved Bugs" report to check if there's a backlog.

Docker

1. What is Docker and why is it useful in QA testing?

Answer:

Docker is an open-source platform that allows you to create, deploy, and run applications inside lightweight, portable containers. Unlike virtual machines (VMs), containers share the host OS kernel but run in isolation, making them faster and less resource-intensive.

In QA, Docker is extremely useful because:

- You can create **identical environments** across dev, QA, and production.
 - Tests can run in isolated containers without affecting the main system.
 - Environment setup becomes **automated and repeatable**, reducing “works on my machine” issues.
 - You can easily test across multiple versions of a service (e.g., MySQL 5.7 vs MySQL 8.0).
 - CI/CD pipelines (Jenkins, GitHub Actions) can run Dockerized test suites reliably.
-

2. Difference between Docker and Virtual Machines for QA?

Answer:

Virtual Machines (VMs):

- Run a full OS with its own kernel → heavier and slower.
- Take minutes to start.
- Consume large amounts of disk and RAM.
- Better for full OS simulation.

Docker Containers:

- Share host OS kernel → lightweight and fast.
- Start in seconds.
- Use fewer resources.
- Ideal for QA automation where **speed & consistency** matter.
- Allows parallel test runs by spinning up multiple containers.

In QA, Docker is preferred because setting up a containerized test environment (like Selenium Grid, API mocks, databases) takes seconds compared to a VM.

3. What is a Docker image and container? How are they different?

Answer:

- **Docker Image:**
A read-only template containing application code, libraries, and dependencies. It's like a "blueprint" for creating containers. For example, an image could be `python:3.10` or `mysql:8.0`.
- **Docker Container:**
A running instance of an image. It's where your application actually runs.

Analogy: Image = recipe, Container = cooked dish.

For QA, you might create an image for your automation framework, then run multiple containers for parallel testing.

4. How do you create and run a Docker container for automation testing?

Answer:

Example: Running a Selenium Grid Node.

```
bash
CopyEdit
docker run -d --name selenium-node -p 4444:4444 selenium/standalone-chrome
```

Steps:

1. Pull image from Docker Hub: `docker pull selenium/standalone-chrome`
2. Run the container with port mapping.
3. Point your test scripts to `http://localhost:4444/wd/hub`.

This approach allows tests to run without needing Chrome installed locally, ensuring **consistent browser versions** for all testers.

5. What is Docker Hub and how do QA engineers use it?

Answer:

Docker Hub is a cloud-based registry for storing and sharing container images.

As a QA engineer, you can:

- Pull ready-made images for databases, browsers, mock servers.
- Push your custom automation framework image so the team can pull and run it easily.
- Use official images for tools like **Postman/Newman**, **Selenium**, or **JMeter** in your pipeline.

6. How to write a Dockerfile for QA automation projects?

Answer:

Example: **Python + Selenium + Chrome**

```
dockerfile
CopyEdit
FROM python:3.10
RUN pip install selenium pytest
COPY tests/ /app/tests
WORKDIR /app/tests
CMD ["pytest", "--html=report.html"]
```

Steps:

1. Start from a base image (python:3.10).
2. Install dependencies.
3. Copy test files.
4. Set working directory.
5. Define default command to run tests.

Benefit: Any team member can run `docker build` & `docker run` to execute tests without environment conflicts.

7. How do you share test environments using Docker Compose?

Answer:

Docker Compose allows you to define multi-container setups in a `docker-compose.yml` file.

Example: Selenium Grid + App Under Test.

```
yaml
CopyEdit
version: '3'
services:
  selenium-hub:
    image: selenium/hub
    ports:
      - "4444:4444"
  chrome:
```

```
image: selenium/node-chrome
depends_on:
  - selenium-hub
environment:
  - HUB_HOST=selenium-hub
```

You can start the whole environment using:

```
bash
CopyEdit
docker-compose up
```

QA benefit: One command sets up browsers, APIs, and databases.

8. What is the difference between `docker run` and `docker exec` in QA testing?

Answer:

- `docker run` → Creates and starts a **new container**.
- `docker exec` → Runs a command **inside an already running container**.

Example for QA:

```
bash
CopyEdit
docker exec -it selenium-node bash
```

This allows you to debug test failures by entering the container without restarting it.

9. How do you persist test results in Docker containers?

Answer:

Containers are **ephemeral**; data is lost when stopped.

Solution: Use **Docker Volumes**.

Example:

```
bash
CopyEdit
docker run -v $(pwd)/reports:/app/reports my-test-image
```

Here, the test reports are saved on the host machine so they're available even after the container stops.

10. How do you optimize Docker images for faster QA pipeline execution?

Answer:

- Use **smaller base images** (e.g., `python:3.10-slim`).

- Combine RUN commands to reduce layers.
- Cache dependencies (pip, npm, Maven).
- Remove unnecessary files after installation.
- Use **multi-stage builds** to separate build/test stages.

Optimized images make CI/CD faster, saving hours in large QA test suites.

11. How do you debug test failures in Docker containers?

Answer:

1. Run the container interactively:

```
bash
CopyEdit
docker run -it my-test-image bash
```

2. Inspect logs:

```
bash
CopyEdit
docker logs <container_id>
```

3. Mount local code directory and modify tests without rebuilding:

```
bash
CopyEdit
docker run -v $(pwd):/app my-test-image
```

This approach allows quick fixes and re-runs.

12. How do you integrate Docker with Jenkins for QA automation?

Answer:

In a Jenkins pipeline:

```
groovy
CopyEdit
pipeline {
    agent { docker { image 'python:3.10' } }
    stages {
        stage('Run Tests') {
            steps {
                sh 'pytest tests/ --html=report.html'
            }
        }
    }
}
```

Benefits:

- Jenkins agents run inside a Docker container with all dependencies.
 - No need to install tools on the Jenkins host.
 - Every build gets a **fresh environment**.
-

13. How can Docker help in cross-browser testing for QA?

Answer:

Using Selenium Grid with Docker:

- Run **Chrome**, **Firefox**, and **Edge** containers simultaneously.
 - Execute the same test suite on multiple browsers in parallel.
 - Containers have predefined versions, avoiding browser update issues.
-

14. What is a Docker Volume and why is it important for QA?

Answer:

A volume is persistent storage managed by Docker.

In QA:

- Use volumes to store logs, screenshots, test reports.
 - Share test data between containers.
 - Keep database state between test runs.
-

15. How do you run database containers for test automation?

Answer:

Example:

```
bash
CopyEdit
docker run -d --name mysql-test -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 mysql:8.0
```

- Your test scripts can connect to `localhost:3306`.
 - Use a preloaded DB dump to simulate real test data.
-

16. What is the difference between `docker ps` and `docker images`?

Answer:

- `docker ps` → Lists running containers.
 - `docker images` → Lists available images.
- In QA debugging, `docker ps -a` is often used to check if a test container exited due to failure.

17. How do you remove unused containers and images to save disk space?

Answer:

```
bash
CopyEdit
docker system prune -a
```

Removes:

- Stopped containers.
 - Unused images.
 - Unused networks.
- Helps in keeping QA systems clean.

18. How do you use Docker for API testing?

Answer:

Example:

- Pull **Postman/Newman** Docker image:

```
bash
CopyEdit
docker run -v $(pwd)/collections:/etc/newman postman/newman run collection.json
```

- This allows API tests to run in CI/CD without installing Postman locally.

19. What are some common Docker commands every QA should know?

Answer:

- `docker pull` → Download image.
- `docker run` → Start container.
- `docker ps` → List containers.
- `docker logs` → View logs.
- `docker exec` → Run commands inside container.
- `docker stop` / `docker rm` → Stop & remove containers.

20. How do you handle environment variables in Docker for QA tests?

Answer:

Use `-e` flag or `.env` files:

```
docker run -e ENV=QA -e API_KEY=12345 my-test-image
```

In your automation code, fetch these environment variables for dynamic test configuration.

Basic CLI & Linux Commands

1. What is the CLI (Command Line Interface) and why should QA engineers know it?

Answer:

The **Command Line Interface (CLI)** is a text-based way to interact with a computer system, instead of using a graphical interface (GUI). QA engineers often work with CI/CD servers, cloud instances, and containerized environments — most of which are Linux-based and accessed via CLI.

Why important for QA:

- **Test environment setup:** Installing tools, libraries, or dependencies.
- **Log analysis:** Searching error patterns in application logs.
- **Automation:** Writing shell scripts to automate repetitive testing tasks.
- **Remote testing:** Accessing test servers via SSH.
- **CI/CD pipelines:** Jenkins, GitHub Actions, GitLab CI often run commands in CLI mode.

Example: If your automated tests fail on a Jenkins server, you may need to SSH into the Linux server and run commands like `tail -f app.log` to investigate.

2. How do you list files and directories in Linux?

Answer:

Use the `ls` command.

```
ls
```

Common options:

- `ls -l` → Long listing with file permissions, owner, size, and timestamp.
- `ls -a` → Includes hidden files (starting with `.`).
- `ls -lh` → Human-readable file sizes (KB, MB).

QA Example: When verifying if a test script or log file exists in a given directory, `ls -l` helps you check last modified times to confirm recent test execution.

3. How do you check your current directory?

Answer:

Use:

`pwd`

Explanation: Prints the full path of the current working directory.

QA Use Case: If you're navigating multiple directories to find logs or config files during debugging, `pwd` confirms your exact location.

4. How do you change directories?

Answer:

Use:

```
cd /path/to/directory
```

Examples:

- `cd /var/log` → Move to system logs folder.
- `cd ..` → Move up one level.
- `cd ~` → Go to your home directory.

QA Example: Moving into the `/opt/jenkins/workspace/` folder to check automation test outputs.

5. How do you create a new directory?

Answer:

Use:

```
mkdir test-reports
```

Explanation: Creates a folder named `test-reports`.

QA Use Case: Before running automated tests, you might create a folder to store reports.

6. How do you create a new file from CLI?

Answer:

Commands:

```
touch file.txt
```

or

```
echo "Hello" > file.txt
```

Explanation: `touch` creates an empty file. `echo` writes content into a file.

QA Example: Creating test data files for manual or automation testing.

7. How do you view the contents of a file?

Answer:

Commands:

```
cat file.txt
less file.txt
more file.txt
```

- `cat` → Displays the entire file.
- `less` / `more` → Displays one screen at a time.

QA Example: Checking automation logs for error messages after test execution.

8. How do you search inside a file for a keyword?

Answer:

Use `grep`:

```
grep "ERROR" app.log
```

Common options:

- `grep -i` → Case-insensitive search.
- `grep -n` → Show line numbers.

QA Example: Quickly finding error lines in test logs to identify failure causes.

9. How do you copy, move, and delete files in Linux?

Answer:

Copy:

```
cp source.txt destination.txt
```

Move/Rename:

```
mv oldname.txt newname.txt
```

Delete:

```
rm file.txt
```

QA Use Case: Copying configuration files to different test environments or renaming logs for archiving.

10. How do you see running processes?

Answer:

Commands:

```
ps aux
top
htop # if installed
```

Explanation:

- `ps aux` → Shows all running processes.
- `top` → Real-time process viewer.

QA Example: Checking if a test server or database service is running.

11. How do you check disk space usage?

Answer:

Command:

```
df -h
```

Explanation: Shows available and used disk space in human-readable format.

QA Example: If tests fail due to "no space left on device" errors, this command helps diagnose it.

12. How do you check memory usage?

Answer:

Command:

```
free -h
```

Explanation: Displays total, used, and free memory.

QA Example: If performance tests fail or run slowly, checking memory usage helps identify bottlenecks.

13. How do you monitor logs in real time?

Answer:

Command:

```
tail -f app.log
```

Explanation: Shows the last lines of a log file and updates live as new entries are added.

QA Example: While running tests, you can watch the log output in real time to detect failures immediately.

14. How do you find a file in Linux?

Answer:

Command:

```
find /path -name "filename.txt"
```

Explanation: Searches the given directory and subdirectories for a file.

QA Example: Finding where Jenkins saved the latest automation report.

15. How do you check network connectivity?

Answer:

```
ping google.com  
curl -I https://example.com
```

- ping → Tests if a server is reachable.
- curl → Fetches a URL's headers or content.

QA Example: Ensuring the application under test is reachable from the test server.

Artificial Intelligence

1. What is Artificial Intelligence (AI) and why is it important for QA?

Answer:

Artificial Intelligence (AI) refers to the ability of machines or systems to perform tasks that typically require human intelligence, such as decision-making, learning, problem-solving, and pattern recognition.

In QA, AI is important because:

- It **reduces manual effort** in test creation and maintenance.
 - It **automatically adapts tests** to UI changes (self-healing).
 - It **predicts defect-prone areas** to optimize testing.
 - It **improves accuracy** in defect detection.
- Example: AI visual testing tools like AppliTools Eyes can detect UI layout shifts that traditional automation may miss.

2. How is AI different from traditional test automation?

Answer:

- **Traditional Automation:** Uses fixed scripts and locators; breaks if the UI changes; fully dependent on pre-written code.
- **AI-based Automation:** Uses ML algorithms to learn from data, adapt to changes, and maintain scripts automatically.
Example: In Selenium, if an element ID changes, a traditional script fails, but AI-powered tools find alternative identifiers to run tests successfully.

3. What is Machine Learning (ML) in QA?

Answer:

Machine Learning is a subset of AI that learns from past data and improves over time without being explicitly programmed.

In QA, ML is used for:

- **Defect prediction**
- **Flaky test detection**
- **Test case prioritization**
Example: ML models can analyze bug history to predict which modules have a higher chance of defects in the next release.

4. What are some AI-powered tools for QA?

Answer:

- **Applitools Eyes** – Visual regression testing.
- **Testim.io** – AI-driven, self-healing automation.
- **Mabl** – Auto-maintenance for functional tests.
- **Functionize** – NLP-based test case creation.
- **Sealights** – Test impact analysis.
- **ReportPortal** – AI log analysis.

5. What is self-healing in AI automation?

Answer:

Self-healing means AI automatically updates locators when the UI changes, preventing test failures.

Example: If a "Login" button ID changes from `btn-login` to `login-btn`, AI can still identify it using other attributes like size, color, or position.

6. How does AI help in defect prediction?

Answer:

AI analyzes historical defect data, code changes, and commit history to predict areas with a higher probability of defects.

Example: If past releases show frequent bugs in the "payment gateway" module, AI will recommend higher test coverage for that area.

7. What is visual testing in AI?

Answer:

Visual testing compares the visual appearance of an application's UI with a baseline image.

AI improves this by detecting **pixel-level changes** while ignoring acceptable differences (e.g., font rendering).

Example: Appliflow Eyes detects if a button's position has shifted slightly, even if the HTML code is the same.

8. How can AI help in exploratory testing?

- Suggest unexplored test scenarios.
 - Record tester actions and convert them to scripts.
 - Highlight rarely used features that might need extra testing.
-

9. What are the challenges of using AI in QA?

- Requires large amounts of clean data.
 - May produce false positives.
 - High tool cost.
 - Steep learning curve for QA engineers.
-

10. How can AI be used in log analysis?

AI tools like **ReportPortal** can scan thousands of log lines to identify failure patterns, group similar errors, and detect root causes.

Example: AI can detect that most failures in a test run come from a database timeout issue.

11. What is AI-based test case prioritization?

Answer:

AI ranks test cases based on risk, historical failures, and recent code changes, ensuring high-risk cases run first.

Example: In a CI/CD pipeline, AI might run regression tests for changed modules before others.

12. How does AI help in test coverage analysis?

Answer:

AI maps requirements, user flows, and code coverage to identify untested areas.

Example: It might suggest adding tests for a newly introduced discount coupon feature that has no existing automated tests.

13. How can AI help in CI/CD pipelines?

- Automates test execution decisions.
- Prioritizes tests based on code changes.
- Predicts release readiness.

Example: Sealights can decide which tests to skip in a CI/CD run to save time.

14. What is the role of NLP in AI QA tools?

Answer:

Natural Language Processing (NLP) allows QA engineers to write test cases in plain English, which the AI converts into executable scripts.

Example: Writing “Click on Login button and verify success message” creates a working test.

15. How can AI handle flaky tests?

Answer:

AI identifies flaky tests by analyzing pass/fail history and execution patterns.

Example: If a test fails 30% of the time without code changes, AI flags it for review.

16. What is predictive analytics in QA?

Answer:

Using AI to forecast possible outcomes based on data trends.

Example: Predicting the number of defects likely in the next sprint based on commit activity.

17. How does AI support performance testing?

- Predict load impact based on historical traffic.
- Identify performance bottlenecks faster.

Example: AI detects that database queries slow down during peak hours and suggests optimization.

18. Can AI generate test data automatically?

Answer:

Yes, AI can create realistic test data by analyzing production datasets while masking sensitive information.

Example: AI generates valid but anonymized user profiles for load testing.

19. How does AI improve regression testing?

Answer:

AI detects which parts of the application are affected by recent changes and runs only relevant regression tests, saving time.

20. How can AI help in root cause analysis?

Answer:

AI scans logs, code changes, and past bug reports to suggest possible causes of failures.

Example: AI might link a UI crash to a specific recent JavaScript library update.

21. What is anomaly detection in QA using AI?

AI detects unusual patterns in application behavior or performance metrics.

Example: AI notices that API response time suddenly doubles, even though functionality is correct.

22. How can AI integrate with Selenium?

- AI can improve element identification (self-healing).
- AI can enhance visual validation.

Example: Tools like Testim integrate AI with Selenium to adapt locators dynamically.

23. What are AI limitations in QA?

- Cannot replace domain knowledge of testers.
 - Needs quality data to be effective.
 - Might miss context-specific issues.
-

24. What is an AI-driven defect management system?

Answer:

A defect management system that uses AI to classify, assign, and prioritize bugs.

Example: AI assigns high-priority defects to the right developer based on expertise.

25. What skills should a QA engineer learn for AI testing?

- Basic Machine Learning concepts.
- Data analysis.
- AI automation tools (Testim, Mabl, AppliTools).
- Python for AI scripting.
- API testing with AI tools.

Tricky & Miscellaneous

1. How do you decide when to stop testing?

Answer:

There is no fixed rule, but testing is typically stopped when:

- **Test Coverage is Sufficient:** All planned test cases are executed.
- **Defect Rate Falls:** New defects found per day drop below an acceptable threshold.
- **Deadlines/Release Date:** Sometimes testing stops due to time constraints.
- **Risk Acceptance:** Business accepts the remaining risks.
- **Pass Criteria Met:** Defined in the Test Plan.

Example:

If you are testing an e-commerce checkout module and all critical P1/P2 bugs are fixed, all high-priority test cases pass, and defect discovery rate is very low, you can stop testing.

2. What is the difference between Verification and Validation?

Answer:

- **Verification:** Ensures the product is being built correctly (process-oriented). Done during development stages.
Example: Reviewing requirement documents.
- **Validation:** Ensures the right product is built (product-oriented). Done after development.
Example: Running functional tests to check if the application meets business needs.

3. What would you do if a developer rejects your defect?

Answer:

- **Recheck the Defect:** Validate with logs, screenshots, or steps to reproduce.
 - **Refer to Requirements:** Show proof from requirement documents or acceptance criteria.
 - **Discuss:** Arrange a quick call with the developer and product owner.
 - **Escalate:** If disagreement persists, involve the QA lead.
-

4. How do you test something without requirements?

Answer:

- Use **Exploratory Testing** techniques.
- Analyze similar modules or past projects.
- Ask stakeholders for high-level goals.
- Use reverse engineering: observe the app and identify possible workflows.

Example:

If you get a payment gateway integration without proper docs, you can explore different payment methods, check response codes, and observe error handling.

5. What is a Severity vs Priority defect?

Answer:

- **Severity:** How badly the defect impacts the system. (Technical impact)
- **Priority:** How soon it needs to be fixed. (Business urgency)

Example:

- *High severity, low priority:* Typo in the “Terms & Conditions” page.
 - *Low severity, high priority:* Logo is incorrect on the home page before a press release.
-

6. How do you test a feature when the environment is unstable?

Answer:

- Use **mock data or stubs** to simulate services.
- Prioritize critical test cases.
- Log environment issues separately from application bugs.

- Use Docker to spin up stable local test setups.
-

7. What would you test in an elevator?

Answer:

Functional: Button operation, floor indicators, emergency stop.

Non-functional: Load capacity, speed, noise, safety compliance.

Negative cases: Pressing multiple buttons, emergency during overload.

8. How do you ensure your test cases are effective?

Answer:

- Cover all **positive, negative, boundary** scenarios.
 - Trace test cases to requirements using **RTM (Requirement Traceability Matrix)**.
 - Regularly update cases after changes.
-

9. How would you test a login page beyond normal credentials?

Answer:

- SQL Injection test.
 - XSS attacks.
 - Empty fields.
 - Special characters in username/password.
 - Multiple failed attempts lockout.
 - Session handling after login.
-

10. How do you test without a UI?

Answer:

- Use API testing with Postman/Newman.
 - Validate database changes directly via SQL queries.
 - Check logs and backend response codes.
-

11. How do you approach testing when timelines are very tight?

Answer:

- Use **Risk-Based Testing**: Focus on critical features.
 - Run **smoke tests** first.
 - Automate repetitive checks.
 - Test in parallel with multiple team members.
-

12. How do you know if your testing is good enough?

Answer:

- Low escape defect rate (few bugs found after release).
 - Positive feedback from stakeholders.
 - High coverage in test metrics.
-

13. Give an example of a bug that is hard to reproduce. How do you handle it?

Answer:

Example: A crash occurs only after long idle time and specific network conditions.

Handling:

- Add logging.
 - Try reproducing in different environments.
 - Use screen recording.
 - Check server logs.
-

14. How do you prioritize test cases?

Answer:

- Based on business impact.
 - Frequency of feature usage.
 - Past defect trends.
 - Critical path first.
-

15. How do you handle intermittent issues?

Answer:

- Run repeated tests.
- Check environmental triggers.
- Log timestamps to match with backend logs.
- Use monitoring tools.

16. What is the difference between Smoke and Sanity testing?

Answer:

- **Smoke Testing:** Broad, shallow testing after a new build. Ensures major functions work.
 - **Sanity Testing:** Narrow, deep testing after a minor fix to check specific functionality.
-

17. How do you deal with changing requirements?

- Maintain flexible test cases.
 - Keep RTM updated.
 - Re-run impacted test cases first.
 - Work closely with the product team.
-

18. What is an Escape Defect?

Answer:

A defect found in production that should have been caught during testing.

Prevention: Improve coverage, peer reviews, better test data.

19. How do you test performance for a web app?

- Use tools like JMeter or Locust.
 - Measure response time, throughput, error rate.
 - Test under load, stress, and endurance conditions.
-

20. What is the most challenging bug you found and how did you debug it?

Answer:

Interviewers expect a **real project example** here.

Example: In a CI/CD pipeline, an API failed only when deployed to staging. Root cause was an outdated SSL certificate in that environment. Debugging involved analyzing Jenkins logs, API responses, and staging configurations.

Real-Time QA Engineer Scenario

1. How would you test an E-commerce website (like Amazon or Flipkart)?

Answer:

Testing an e-commerce site involves multiple functional and non-functional scenarios.

I would divide testing into modules:

- **Homepage:** Check navigation, banners, responsiveness, broken links.
- **Search:** Verify product search with valid/invalid keywords, filters, sorting, spelling mistakes.
- **Product Details Page:** Check images, zoom, availability, add to cart, wishlist.
- **Cart & Checkout:** Add/remove products, apply coupons, quantity update, price recalculation.
- **Payment Gateway:** Test multiple methods (UPI, Credit card, COD), retry on failure, refund process.
- **Order Management:** Verify order status (Pending → Shipped → Delivered), cancellation/refund flow.
- **Email/SMS Notifications:** Confirm that order confirmation, shipping updates are sent.

👉 Example:

If I'm testing "**Apply Coupon Code**", I'd check valid coupon (discount applied), expired coupon (error shown), and invalid coupon (error shown).

This ensures **end-to-end functional flow + usability + payment reliability**.

2. How many test cases will you write to test an E-commerce site?

Answer:

There's no fixed number — it depends on scope. For a medium e-commerce site:

- Homepage → 20–30 test cases
- Login/Register → 30–40 test cases
- Search & Filter → 40–50 test cases
- Cart & Checkout → 50–70 test cases
- Payments → 70–100 test cases
- Order Management → 40–60 test cases

👉 Roughly **300–400 test cases for full regression**.

For **smoke testing**, I'd prioritize ~30–40 key cases (login, search, add to cart, checkout, payment).

3. What if an API request passes but UI fails (e.g., Order API says "Success" but UI shows "Failed")?

Answer:

This happens due to **UI and backend not being in sync**.

Steps:

1. Validate API logs → Check if API responded with 200 success.
2. Inspect UI logs (browser console, network tab) → Look for errors (JS, timeout, incorrect response handling).
3. Verify mapping → Maybe API field changed but UI is still reading old key.

👉 Example:

API response: { "orderStatus": "success" }

But UI expects status = success. → Misalignment.

As a tester, I'd raise a **defect for integration mismatch** and attach:

- API response proof
- UI screenshot
- Console logs

4. If a bug is not accepted by developers, how do you convince them?

Answer:

I follow a professional approach:

1. **Evidence-based communication** → Provide logs, screenshots, API responses, steps to reproduce.
2. **Business impact** → Explain how it affects customer experience.
Example: "If user cannot apply a coupon, company may lose sales."
3. **Data-driven approach** → Compare behavior with acceptance criteria (user story, requirement doc).

👉 Example:

If developer says "This is not a bug, it's by design," I'd show **requirement doc** or **mockup from BA** to prove mismatch.

If still not accepted → escalate to QA Lead / Product Owner.

5. What if your Pull Request (PR) is not approved for merging? How will you handle it?

Answer:

- First, I'd **check reviewer's comments** → maybe I missed code quality checks, proper test coverage, or naming conventions.
- If I strongly believe my code is correct, I'd schedule a **quick call with reviewer** to walk through code.
- I'd use **facts** (test results, logs, automated run success) instead of arguments.
- If still stuck → escalate politely to Lead/Manager.

👉 Example:

If PR contains Selenium automation scripts for login tests, and reviewer says “Not optimized,” I’d show proof of execution time & coverage, then ask if they prefer Page Factory or different approach.

6. How do you test “Add to Cart” functionality in an e-commerce app?

Answer:

- Add single product → Verify cart updates correctly.
- Add multiple quantities → Price & total calculation correct.
- Remove item → Cart count updates.
- Empty cart → Verify checkout button disabled.
- Cross-browser/device → Cart persists on login.

👉 Example:

Bug found in real projects: **Cart quantity didn’t update when user pressed back button.**

7. How do you handle a situation where high-severity bug is found just before release?

Answer:

- Immediately **inform QA Lead, Dev Lead, Product Owner.**
- Show **business impact** (e.g., payment gateway not working = revenue loss).
- Suggest **workarounds** if available.
- If fix possible → delay release slightly and retest.
- If no fix → request business decision (release with known bug or block release).

👉 Example:

Found bug: “COD not available at checkout.” → I’d escalate since it blocks 30% of orders.

8. How do you test API rate limiting (throttling)?

Answer:

- Send multiple requests (100/sec).
- Verify correct response code (429 Too Many Requests).
- Check if retry after “X seconds” header is present.

👉 Example:

In payments API, system allows only 5 retries → test by sending 6th request → must fail.

9. How do you test if email notifications are sent after order confirmation?

Answer:

- Place order → Verify email received.
 - Validate subject, content, order ID, timestamp.
 - Test negative case: Fake email ID → Should fail gracefully.
 - Test bulk order → Multiple emails sent correctly.
-

10. How do you ensure your automation scripts are stable and not flaky?

Answer:

- Use **Explicit waits** instead of hard sleeps.
- Apply **Page Object Model (POM)** for reusability.
- Handle dynamic locators with CSS/XPath strategies.
- Run scripts in CI/CD (Jenkins) for consistency.

👉 Example:

Login test failing sometimes → Added wait for “Login button enabled.” Bug fixed.

11. If production issue is reported but you didn't catch it in testing, how do you handle it?

Answer:

- Accept responsibility & investigate root cause.
- Check **test coverage** → was scenario missed?
- Improve test cases / automation suite.
- Add regression suite to prevent reoccurrence.

👉 Example:

Missed bug: “Coupon not working with guest checkout.” → Added guest checkout test cases later.

12. How do you test multi-currency payment in an e-commerce site?

Answer:

- Select USD, INR, EUR → Check conversion rate, rounding.
 - Test different gateways (PayPal, Stripe).
 - Refund in same currency.
 - Validate against live exchange rates (mock data for testing).
-

13. How do you test search functionality with filters (price, brand, category)?

Answer:

- Search valid product → Filter by brand, check results.
- Search invalid product → Show “No results.”
- Apply multiple filters → Results narrow down correctly.
- Remove filters → Results reset.

👉 Example:

Bug found: Search worked, but **filters applied only after refreshing page**.

14. If two developers raise bugs on each other's code, how do you as QA handle it?

Answer:

- Stay neutral → Focus on **requirement vs actual behavior**.
 - Retest both scenarios with documentation.
 - Attach logs/screenshots → Final decision based on facts.
-

15. How do you test mobile responsiveness of an e-commerce site?

Answer:

- Test on Chrome DevTools (mobile view).
 - Verify layout on iOS/Android devices.
 - Test touch actions (swipe, pinch zoom).
 - Check if checkout works smoothly on mobile.
-

16. How do you handle a failed automation build in Jenkins?

Answer:

- Check console logs.
 - Identify if failure due to **environment, script, or actual bug**.
 - If script issue → Fix and rerun.
 - If environment → Restart server/test DB.
 - If real bug → Log in JIRA.
-

17. How do you prioritize test cases when release deadline is short?

Answer:

- Run **Smoke Test** → Critical flows (Login, Search, Checkout, Payment).
 - Run **Regression only on impacted modules**.
 - Use **Risk-based testing** → High impact areas tested first.
-

18. How do you test refund flow in e-commerce?

Answer:

- Place an order → Cancel → Verify refund initiated.
 - Refund via wallet, card, UPI → Different timelines.
 - Partial refund (1 item out of 3).
 - Invalid refund request → Show error.
-

19. How do you test when multiple users access same product stock?

Answer:

- Simulate 2 users ordering same last item.
 - Ensure one order succeeds, other gets “Out of stock.”
 - Prevent overselling (negative stock).
-

20. How do you test security in login (e.g., brute force, SQL injection)?

Answer:

- Enter wrong password 5 times → Account locked.
 - SQL injection attempt in username → Should sanitize input.
 - Check OTP/password encryption.
-

21. How do you test user registration with email and mobile number?

Answer:

I'd validate both positive and negative cases:

- **Positive:**
 - Register with valid email & strong password → success message.
 - OTP received for mobile/email verification.

- **Negative:**
 - Invalid email format (abc@com) → Error message.
 - Already registered email/mobile → Duplicate error.
 - Weak password (12345) → Rejected.

👉 Example:

Bug I faced: Registration accepted duplicate mobile numbers across accounts → escalated as **data integrity issue**.

22. How do you test "Forgot Password" functionality?

Answer:

- Enter valid registered email → OTP/Reset link sent.
- Enter unregistered email → Show “Email not found.”
- Click reset link → New password setup.
- Expired link → Show “Link expired.”

👉 Example:

Real bug: Password reset link expired after **2 minutes instead of 24 hours** → blocked many users.

23. How do you test file upload functionality (like profile picture upload)?

Answer:

- Upload valid JPG, PNG → Success.
- Upload large file (> limit, e.g., 5MB) → Error.
- Upload malicious file (.exe) → Blocked.
- Upload duplicate → Replaces old one.

👉 Example:

Bug: Upload worked on Chrome but failed on Safari due to **CORS issue**.

24. How do you test payment refund timeline?

Answer:

- Wallet refund → Instant.
- Credit card refund → 5–7 days.
- UPI refund → 24 hours.
- Cancel order → Verify refund initiation timestamp logged.

👉 Example:

In one project, refund marked as “**Processed**” in DB but didn’t reflect in UI → raised bug for **UI inconsistency**.

25. How do you test a shopping cart when multiple tabs are open?

Answer:

- Add item in Tab 1 → Check reflected in Tab 2.
- Remove item in Tab 2 → Updated in Tab 1.
- Checkout in Tab 1 while Tab 2 still open → No duplicate order.

👉 Example:

Bug: Adding product in Tab 1 didn’t update Tab 2 until **page refresh** → inconsistent cart state.

26. How do you test search suggestions (auto-complete)?

Answer:

- Typing “Lap” → Suggest “Laptop, Laptop Bag.”
- Wrong spelling → Fuzzy match (e.g., “Laptoop”).
- Empty search → No suggestions.
- SQL injection attempt → Sanitized.

👉 Example:

Bug: Auto-suggestions leaked **out-of-stock products** that should’ve been hidden.

27. How do you handle if a developer says "It's working on my machine"?

Answer:

- Reproduce on QA environment → record video/logs.
- Compare versions of build, DB, config.
- Highlight **environment mismatch**.
- If still denied → Involve DevOps to check deployment consistency.

👉 Example:

Bug in checkout → worked locally but failed in staging → root cause was **different API endpoint configs**.

28. How do you test "Add to Wishlist" functionality?

Answer:

- Add single item → Wishlist updated.
- Remove item → Removed.
- Add out-of-stock item → Still allowed in wishlist.
- Wishlist persists across sessions.

👉 Example:

Bug: Wishlist cleared automatically after logout → unacceptable for customers.

29. How do you test multi-user login sessions (same account on different devices)?

Answer:

- Login on Device A & Device B simultaneously.
- Add to cart on A → Syncs to B.
- Logout from one device → Should not force logout on all (unless security setting requires).

👉 Example:

Bug: One login invalidated the other → users couldn't shop across devices.

30. How do you test performance of product search API?

Answer:

- Check response time → Must be < 1 sec for 100 products.
- Load test → 5000 concurrent users searching.
- Stress test → 20,000 requests → API should fail gracefully.

👉 Example:

Observed search API slowed to **6s under 1000 users** → flagged for optimization.

31. How do you test cross-browser compatibility?

Answer:

- Test on Chrome, Firefox, Safari, Edge.
- Verify CSS, fonts, alignments.
- Check JavaScript functions (dropdowns, modals).
- Automate via **Selenium Grid / BrowserStack**.

👉 Example:

Bug: “Checkout button not clickable on Safari” due to CSS overlay issue.

32. How do you test product price calculation (discounts, tax, shipping)?

Answer:

- Apply % discount coupon → Correct calculation.
- Apply fixed discount (₹100 off).
- Apply tax → Correct GST calculation.
- Add shipping charge.
- Edge case: 100% discount coupon.

👉 Example:

Bug: Discount applied twice when coupon re-entered → double deduction.

33. How do you test API timeout handling?

Answer:

- Delay API response > configured timeout (e.g., 30s).
- Verify UI shows “Something went wrong” not infinite loader.
- Retry logic implemented.

👉 Example:

Bug: App kept spinning for 5 mins because timeout not handled.

34. How do you handle priority vs severity conflicts in bugs?

Answer:

- **Severity** = impact on system.
- **Priority** = urgency of fix.

👉 Example:

- Spelling mistake in homepage → Low severity, High priority (customer-facing).
- Crash in admin panel → High severity, Low priority (internal users only).

As QA, I'd align with **Product Owner** for business impact.

35. How do you test order cancellation flow?

Answer:

- Cancel before shipment → Allowed.
- Cancel after shipment → Not allowed.
- Refund initiated after cancellation.
- Multiple cancel attempts → Only first valid.

👉 Example:

Bug: Cancellation allowed even after “Out for delivery” status.

36. How do you test “Apply Promo Code” functionality?

Answer:

- Valid promo → Discount applied.
- Expired promo → Error.
- Invalid promo → Error.
- Promo applied with minimum cart value condition.

👉 Example:

Bug: Promo code applied even when cart < minimum value.

37. How do you test order tracking system?

Answer:

- Order status → Pending → Shipped → Out for Delivery → Delivered.
- Wrong tracking ID → Error message.
- Delivery delay → ETA updated.
- Email/SMS notifications sent.

👉 Example:

Bug: Tracking page showed “Delivered” but package still in transit.

38. How do you test login with social media (Google, Facebook)?

Answer:

- Valid login → Redirect to homepage.
- Invalid credentials → Error from OAuth.
- Revoke permission → Login fails.

- Multiple accounts → User chooses correct one.

👉 Example:

Bug: Google login worked but didn't fetch user profile → incomplete onboarding.

39. How do you test checkout process with multiple payment methods?

Answer:

- COD, UPI, Credit/Debit Card, Wallet.
- Card with insufficient funds → Declined.
- UPI with expired VPA → Error.
- Wallet balance + card combined.

👉 Example:

Bug: UPI payments succeeded but UI still showed "Payment Failed."

40. How do you test caching in e-commerce websites?

Answer:

- Add product → Refresh page → Updated stock must show.
- Clear cache → Latest data fetched.
- CDN caching for images.
- Ensure sensitive data (cart, payment) not cached.

👉 Example:

Bug: Out-of-stock product still showing available due to stale cache.

41. How do you test if an API response schema changes?

Answer:

- Maintain schema validation in automation (JSON schema).
- If API previously returned { "id": 123 } and now returns { "userId": 123 }, UI might break.
- Compare API contract against Swagger/Postman docs.

👉 Example:

In one project, developer renamed `orderStatus` → `status` → UI crashed. Caught via schema validation test.

42. How do you test database data integrity after placing an order?

Answer:

- Place order → Verify `Orders` table has correct details (orderId, amount, status).
- `Inventory` table should reduce stock.
- `Payment` table should log transaction.
- No orphan records (orders without payment).

👉 Example:

Bug: Inventory was not reduced after COD orders → stock showed wrong availability.

43. How do you test API error codes?

Answer:

- **200** → Success.
- **400** → Bad Request.
- **401** → Unauthorized.
- **404** → Resource not found.
- **500** → Internal server error.

👉 Example:

Payment API returned **200 even on failure** → UI showed “Success” incorrectly. Reported as **critical bug**.

44. How do you test database rollback if a transaction fails?

Answer:

- Simulate payment failure → Order should not be created in DB.
- Partial rollback check → No half-complete records.

👉 Example:

Bug: Payment failed but order row still created → users got “ghost orders.”

45. How do you test API pagination?

Answer:

- Fetch products with `limit=10, offset=0`.
- Next call → `offset=10`.
- Verify no duplicates, no missing data.
- Edge case: last page with `< 10` results.

👉 Example:

Bug: API returned duplicate products across pages.

46. How do you test automation scripts in parallel execution?

Answer:

- Run 10 login tests simultaneously using Selenium Grid.
- Ensure data isolation (unique test data per thread).
- Avoid static variables/shared drivers.

👉 Example:

Parallel checkout tests failed because multiple threads used same account → fixed by creating **separate test data pools**.

47. How do you test CI/CD pipeline for automation tests?

Answer:

- Verify Jenkins job triggers on code push.
- Smoke tests run first, regression later.
- Report published (Allure/Extent).
- Slack/email notifications configured.

👉 Example:

Bug: Jenkins marked build “green” even when 10% tests failed due to **incorrect exit code handling**.

48. How do you test if API caching is implemented correctly?

Answer:

- Call same API twice → First call slower, second call faster (cache hit).
- Invalidate cache → New data fetched.
- Sensitive APIs (balance check) → Should never cache.

👉 Example:

Bug: Cart API cached → User saw old cart even after removing items.

49. How do you test SQL injection vulnerability?

Answer:

- Enter input like ' OR 1=1 -- in login field.
- If query not sanitized → All users can log in.

- Verify error messages don't leak DB info.

👉 Example:

Tested login with `abc' --` → bypassed authentication. Reported as **P1 Security Bug**.

50. How do you test automation framework stability?

Answer:

- Run suite multiple times → Should pass consistently.
- Retry logic for flaky tests.
- Centralized test data.
- Proper exception handling.

👉 Example:

Flaky login test failed randomly → fixed by adding **explicit wait** for login button visibility.

51. How do you test API authentication (OAuth, JWT)?

Answer:

- Valid token → API success.
- Expired token → 401 Unauthorized.
- Tampered token → Invalid signature.
- No token → Blocked.

👉 Example:

Bug: API accepted expired JWT tokens → **serious security issue**.

52. How do you test CI/CD rollback mechanism?

Answer:

- Deploy faulty build → Pipeline should rollback to last stable version.
- Verify rollback logs and notifications.

👉 Example:

During a faulty checkout deployment, rollback didn't trigger → site went down for 30 mins. Reported as **pipeline gap**.

53. How do you test if automation test data is re-usable?

Answer:

- Ensure test data cleanup after execution.
- Use random generators for email, order ID.
- Reset DB or use mock services.

👉 Example:

Tests failed because same **user email** reused → “Email already registered” error. Fixed with **UUID-based email creation**.

54. How do you test API concurrency issues?

Answer:

- Trigger 10 parallel API calls to order same product.
- Verify only one succeeds, others get “Out of stock.”
- No duplicate records.

👉 Example:

Bug: Two users placed order for last item → inventory went negative.

55. How do you test Jenkins pipeline scheduling?

Answer:

- Verify nightly regression triggers at 2 AM.
- Check logs to confirm schedule execution.
- Failures → Notifications sent to Slack/email.

👉 Example:

Bug: Scheduled job skipped on weekends due to **cron misconfiguration**.

56. How do you test data migration when moving from old DB to new DB?

Answer:

- Row counts before & after migration.
- Data consistency (no null values).
- Key relationships maintained.
- Random record checks.

👉 Example:

Bug: OrderHistory table lost foreign key mapping after migration.

57. How do you test performance of automation framework?

Answer:

- Measure total suite execution time.
- Identify slowest tests.
- Optimize locators, reduce waits, parallelize execution.

👉 Example:

Regression suite took 6 hours → reduced to 1.5 hours using **parallel execution + optimized waits**.

58. How do you test data encryption in API/DB?

Answer:

- API should return masked data (e.g., ****1234 for card).
- DB should store encrypted (AES/Hash), not plain text.
- Verify secure transmission (HTTPS).

👉 Example:

Bug: API returned full **credit card number** in response → escalated as **Critical P0 issue**.

59. How do you test log generation in applications?

Answer:

- Perform actions → Verify logs created.
- Error case → Error logs recorded.
- Logs rotated daily to prevent overflow.

👉 Example:

Bug: Application logs exposed **user passwords** in plain text.

60. How do you test rollback in automation framework if test fails?

Answer:

- Test fails → Rollback test data.

- Example: Create order test → If fails, order must be deleted.
- Implement cleanup in @AfterMethod.

👉 Example:

Order creation test failed → but leftover orders stayed in DB → polluted future tests. Fixed with **cleanup method**.

⚡ Vamsi, that's **Questions 41–60 with long answers, real project examples, and deep explanations.**

Would you like me to make the **next 20 (61–80)** around **Agile/Scrum, Bug lifecycle, Stakeholder communication, and tricky “convince-the-developer” type real-time scenarios?**

You said:

yes

ChatGPT said:

Awesome 🔥 Now let's go into **Agile/Scrum, Bug Lifecycle, Stakeholder communication, and tricky “convince-the-developer” type scenarios** — the kind that most **QA / Test Engineer / Automation Engineer** interviews love to ask, because they test both **technical + communication skills**.

Here's the next batch:

✅ Real-Time QA Scenario Questions (61–80)

61. In Agile, what will you do if developers deliver a user story very late in the sprint?

Answer:

- If testing time is very short, I'd:
 - Prioritize smoke tests → Validate critical flow.
 - Automate regression in parallel.
 - Highlight risk in the **daily stand-up** to Scrum Master & PO.
- If critical bugs found → Escalate risk of sprint goal failure.

👉 Example:

In one sprint, developer delivered “Apply Promo Code” story on the last day → only smoke tested discount logic, deferred full boundary tests to next sprint.

62. How do you test if Acceptance Criteria (AC) is unclear in a story?

Answer:

- Ask **clarifying questions** in grooming/refinement.
- Use **3 Amigos approach** (Dev, QA, PO).
- Document assumptions if PO not available.

👉 Example:

AC said: “User should see discounts.” I asked → *Flat discount or percentage? Applied before or after tax?* → Found missing business rules.

63. If you find a bug during sprint demo, how do you handle it?

Answer:

- Don’t hide → Show bug transparently.
- Explain **expected vs actual behavior**.
- Raise Jira bug → Link to story.
- If minor UI → Fix in next sprint.
- If critical → Block release until fixed.

👉 Example:

During demo, coupon field accepted invalid characters → demo failed but stakeholders appreciated honesty.

64. How do you handle a developer who always rejects your bugs as “Not a bug”?

Answer:

- Provide **step-by-step repro, screenshots, logs**.
- Compare with **requirements/AC**.
- If conflict → involve PO/BA for decision.
- Maintain professional tone, avoid arguments.

👉 Example:

Developer rejected “Cart not updating on back button.” I showed requirement doc → bug got accepted.

65. How do you test a user story that has dependencies on another team?

Answer:

- Identify dependency early in **sprint planning**.
- Use **mock APIs or stubs** if other team not ready.

- Blocker tickets raised in Jira.
- Sync in **Scrum of Scrums** if multi-team project.

👉 Example:

Payment service not ready → I mocked API response in Postman to test checkout flow.

66. If PO insists on releasing with a known bug, what will you do?

Answer:

- Clearly explain **business impact**.
- Provide examples: “User can’t apply coupon → revenue loss.”
- If still insists → Document as **Known Issue** in release notes.
- Add bug to backlog for next sprint.

👉 Example:

Released with known issue “Wishlist not persisting” → communicated upfront → reduced customer complaints.

67. How do you estimate testing effort in Agile?

Answer:

- Use **Story Points** or hours.
- Consider: complexity, test data, environments, automation scope.
- Example scale:
 - Simple login → 2 points.
 - Payment gateway integration → 8 points.

👉 Example:

Estimated “Refund flow” as 5 points because of multiple scenarios (full, partial, wallet, card).

68. How do you handle a blocker bug in the middle of sprint?

Answer:

- Raise immediately in stand-up.
- Create **blocker Jira ticket**.
- Collaborate with dev → maybe test other non-dependent stories.
- Escalate if no quick resolution.

👉 Example:

Checkout API down → blocked entire cart testing. Escalated to DevOps, switched to regression testing meanwhile.

69. How do you test when requirements keep changing?

Answer:

- Follow Agile principle → embrace change.
- Use modular test cases → easy to update.
- Automate stable flows.
- Track changes in **Jira Confluence**.

👉 Example:

Requirement: “COD enabled for all” → changed to “COD only for orders < ₹2000.” Updated test cases quickly.

70. How do you report testing progress in Agile?

Answer:

- Daily stand-up → blockers, completed tasks, next tasks.
- Jira dashboard → Story status (To Do, In Progress, Done).
- Share **Test Coverage Report** (executed vs pending).

👉 Example:

I used Jira Xray → showed 75% test execution, 5 open bugs → gave PO confidence before release.

71. If you miss testing a scenario and it fails in production, how do you handle it?

Answer:

- Accept mistake, no blame game.
- Do **root cause analysis** → why missed? (requirement gap, lack of test data).
- Add to regression suite.
- Improve process to avoid repeat.

👉 Example:

Missed “Guest checkout + coupon” → caused prod bug. Added guest checkout tests permanently.

72. How do you convince management to give more time for testing?

Answer:

- Show risk matrix → bugs found vs time spent.
- Provide defect leakage stats → “Less testing = higher production issues.”
- Suggest parallel automation to save time.

👉 Example:

Convincing PO to give **1 more day** found 3 more P1 bugs → proved value of extended testing.

73. How do you test bug fixes quickly in Agile?

Answer:

- Retest bug → verify exact scenario.
- Run related regression cases.
- Automate if recurring.

👉 Example:

Fix for “Quantity not updating in cart” → retested with multiple products + checkout.

74. How do you test a feature with no documentation?

Answer:

- Explore feature using **Exploratory Testing**.
- Discuss with dev/PO for expectations.
- Check similar existing features.
- Document findings as you test.

👉 Example:

Feature “Gift wrapping” had no doc → tested by trial, confirmed with dev, wrote own test notes.

75. How do you manage test cases in Agile (where requirements evolve)?

Answer:

- Keep **lightweight test cases** in Jira/Xray/Zephyr.
- Use **checklists** for evolving features.
- Automate repetitive ones.

👉 Example:

Maintained “Search & Filter” checklist instead of 50 detailed cases → saved time when filters changed often.

76. How do you test sprint deliverables in very short time?

Answer:

- Apply **Risk-based testing**.
- Focus on **high-priority user journeys** (login, checkout, payment).
- Automate regression in background.

👉 Example:

Had 4 hours to test → covered login, add to cart, payment only. Reported 1 critical bug → sprint saved.

77. How do you handle conflict with a developer?

Answer:

- Focus on **data, not emotions**.
- Show logs, steps, screenshots.
- If disagreement persists → escalate to Lead/PO.
- Keep collaborative tone.

👉 Example:

Developer said “Bug not valid.” I demonstrated in staging → he accepted politely.

78. How do you test Agile user stories that don’t fit in one sprint?

Answer:

- Split into smaller deliverables.
- Example: Story “Refund system” →
 - Sprint 1: Refund initiation API.
 - Sprint 2: UI & email confirmation.
- Test incrementally.

👉 Example:

Refund flow was too large → we split into API + UI → tested separately.

79. How do you test when development and QA environments differ?

Answer:

- Verify configs with DevOps.
- Use same DB seed/data set.
- Raise env parity issue if major gaps.

👉 Example:

Bug appeared in QA but not dev → root cause: **different DB versions**.

80. How do you test when multiple teams deliver microservices in parallel?

Answer:

- Test each microservice with **mock contracts**.
- Once integrated, run **end-to-end flows**.
- Automate contract tests with Postman/Newman.

👉 Example:

Checkout used **Payment + Inventory microservices**. Payment team delivered late → I mocked payment API to continue testing.

81. How will you test cart abandonment recovery in an e-commerce site?

👉 Answer:

- Add items to cart → abandon checkout.
 - Verify system sends email/SMS/push notification after X hours.
 - Ensure coupon code/discount is applied correctly.
 - Test API trigger for abandoned cart.
 - **Example:** User adds shoes worth ₹2000 but leaves. After 24 hrs, system sends a mail: *“Complete purchase & get 10% off”*. Verify link redirects to pre-filled cart.
-

82. How will you test price updates when a product goes on sale?

👉 Answer:

- Verify backend price update in DB.
- UI should reflect sale price everywhere (product page, cart, order summary, invoice).
- Ensure old customers don't get wrong prices.
- **Example:** Product price drops from ₹1500 → ₹1200. If a user adds to cart before discount but buys later, order should calculate new discounted price.

83. How do you ensure test cases cover edge scenarios for payment gateway?

👉 **Answer:**

- Test expired card, insufficient balance, wrong CVV, network failure, retry option.
- Verify duplicate payment handling (should not double charge).
- **Example:** API returns "Transaction Declined - 51" (insufficient funds), UI should show *"Payment Failed: Low Balance"*.

84. If a stakeholder rejects your bug saying "Not reproducible", how will you handle?

👉 **Answer:**

- Provide **detailed steps**, screenshots, and logs.
- Record a **video proof**.
- Check **environment differences** (e.g., staging vs prod).
- **Example:** You find "Cart not updating in Safari browser" but Dev says it works on Chrome. You share a video + browser logs to prove.

85. How do you test an OTP-based login system?

👉 **Answer:**

- Validate OTP is generated correctly.
- Expiry time (e.g., 3 mins).
- Wrong OTP → proper error message.
- Retry & resend functionality.
- **Example:** If OTP expires in 3 mins but system accepts after 5 mins → security bug.

86. What if API response is slow, but UI has no timeout handling?

👉 **Answer:**

- API SLA should be $\leq 2s$.
- UI should show a loader & timeout error after X secs.
- **Example:** Payment API takes 30s, but UI keeps spinning forever → poor UX. Raise a bug for **timeout handling**.

87. How will you test localization of an e-commerce site (multi-language)?

👉 Answer:

- Switch language → verify translations across pages.
 - Currency conversion (₹ → \$, €, etc.).
 - RTL (Right-to-Left) for Arabic.
 - **Example:** In German site, “Add to Cart” should be “In den Warenkorb”. If only half-translated → bug.
-

88. How do you test performance of bulk order processing?

👉 Answer:

- Place 100–1000 items in cart.
 - Validate system doesn’t slow down.
 - API batch requests handling.
 - **Example:** If 500 items take 2 minutes to process, log a performance bug.
-

89. If Dev doesn’t approve your PR merge for automation, how do you convince?

👉 Answer:

- Share code quality checks (linting, SonarQube).
 - Provide test execution proof.
 - Show how automation reduces regression time.
 - **Example:** If manual checkout testing takes 1 hr, but your automation runs in 3 mins, highlight ROI.
-

90. How do you test coupons and discount codes?

👉 Answer:

- Apply valid coupon → discount applied.
 - Expired coupon → error.
 - Min cart value validation.
 - One-time-use coupon check.
 - **Example:** If coupon "NEW100" is valid only for new users but works for existing ones, it’s a **business loss bug**.
-

91. How do you test concurrency when multiple users update cart at same time?

👉 Answer:

- Two users try to buy **last item** simultaneously.
 - Ensure only one order succeeds.
 - Locking mechanism check in DB.
 - **Example:** If both users purchase last laptop → overselling → **critical bug**.
-

92. How do you ensure data consistency across UI, API, DB?

👉 Answer:

- UI total = API response total = DB record.
 - Sync issues validation.
 - **Example:** Order shows ₹2000 in UI but API says ₹2200 → mismatch.
-

93. How do you test wishlist feature in e-commerce?

👉 Answer:

- Add/remove items.
 - Persist wishlist after logout/login.
 - Cross-device sync.
 - **Example:** Add 5 items on mobile → login in web → items should still be present.
-

94. How do you handle flakiness in automation scripts?

👉 Answer:

- Add proper waits (explicit wait > thread.sleep).
 - Stabilize locators.
 - Retry failed tests.
 - **Example:** If “Place Order” button loads late, script fails randomly. Add WebDriverWait condition.
-

95. How will you test email notifications for order confirmation?

👉 Answer:

- Trigger order → check mail delivery.
 - Verify subject, body, order ID.
 - Validate links inside mail.
 - **Example:** Order confirmation mail missing order ID → bug.
-

96. What if business says a bug is minor but you think it's critical?

👉 Answer:

- Highlight **impact on user & revenue**.
 - Give real-world scenario.
 - **Example:** *"Add to Cart button misaligned"* may look minor, but if unclickable in iPhone Safari → critical.
-

97. How do you test order cancellation?

👉 Answer:

- Cancel before shipment → allowed.
 - Cancel after shipment → not allowed.
 - Refund initiation check.
 - **Example:** If system allows cancellation after delivery → bug.
-

98. How do you test search functionality?

👉 Answer:

- Search by product name, partial text, SKU, category.
 - Validate ranking (relevance).
 - Handle typos/autocomplete.
 - **Example:** Searching "iphon" should still suggest "iPhone".
-

99. How do you test stock management in inventory system?

👉 Answer:

- Buy last item → stock should reduce.
- API should update DB correctly.
- Out-of-stock item shouldn't be purchasable.
- **Example:** If 10 items in stock but 15 orders placed → overselling bug.

100. How do you test a new feature rollout using A/B testing?

👉 Answer:

- 50% users see old flow, 50% see new flow.
- Validate segmentation logic.
- Ensure tracking metrics are logged.
- **Example:** If discount banner is shown only to group A but group B also sees → rollout bug.

-----THE END-----