Q1: What is manual testing?

A1:

Manual testing involves executing test cases without the use of automation tools. Testers manually perform the steps, validate the system's behavior, and report issues if the software does not meet the expected outcomes.

Q2: What are the different types of manual testing?

A2:

- **Functional Testing:** Validates the functionality of the software based on requirements.
- **Non-functional Testing:** Validates the non-functional aspects such as performance, usability, and security.
- **Regression Testing:** Ensures that new changes do not break existing functionality.
- **Smoke Testing:** Verifies that the basic functionalities of the application work after a new build.
- **Sanity Testing:** Verifies that specific functionalities work after minor changes.
- **Exploratory Testing:** Testers explore the software to identify defects without predefined test cases.
- **User Acceptance Testing (UAT):** Ensures that the software meets business requirements.

Q3: What is the difference between verification and validation?

A3:

- **Verification:** Ensures the product is being built correctly (check if the product is being developed according to specifications).
- **Validation:** Ensures the product meets the user's needs (check if the product satisfies the requirements).

Q4: What is the difference between a test case and a test scenario?

A4:

- **Test Case:** A set of conditions under which a tester evaluates a system or component. It is more detailed and has specific input, actions, and expected results.
- **Test Scenario:** A high-level concept that defines a particular testable situation, which may consist of multiple test cases.

Q5: What are the key components of a test case?

A5:

- **Test Case ID:** Unique identifier.
- **Test Case Name:** Descriptive name.
- **Pre-conditions:** Initial conditions required before test execution.
- **Test Steps:** Detailed instructions to execute.
- **Test Data:** Input values for execution.
- **Expected Result:** Expected behavior after test execution.
- **Actual Result:** Actual behavior after execution.
- **Pass/Fail:** Outcome of the test case.
- **Post-conditions:** State after test execution.

Q6: What do you mean by test data? How do you generate it for manual testing?

A6:

Test data is the input used to validate the behavior of an application during testing. It can be generated based on the application's functionality, use cases, and data models. Test data can be created manually, taken from production systems, or generated using tools for large datasets.

Q7: What is a bug life cycle?

A7:

The bug life cycle includes several stages, such as:

- **New:** Bug is identified and logged.
- **Assigned:** Assigned to a developer for fixing.
- **Open:** Developer starts working on the bug.
- **Fixed:** Developer fixes the bug.
- **Retested:** Tester verifies the fix.

- **Closed:** If the issue is resolved, the bug is closed.
- **Reopened:** If the issue persists, the bug is reopened.

Q8: How do you write a test case for a login page?

A8:

Test case for login page:

- **Test Case ID:** TC_Login_001
- **Test Case Name:** Validate successful login with valid credentials
- **Pre-conditions:** User must have a valid username and password.
- **Test Steps:**
    - Open the login page.
    - Enter a valid username and password.
    - Click the login button.
- **Expected Result:** User is logged in and redirected to the homepage.
- **Actual Result:** [To be filled after test execution]
- **Pass/Fail:** [To be filled after test execution]

Q9: What is exploratory testing?

A9:

Exploratory testing involves simultaneous learning, test design, and execution. The tester explores the application and its features to discover defects without predefined test cases, relying on creativity, experience, and intuition.

Q10: What is the difference between positive and negative testing?

A10:

- **Positive Testing:** Verifies that the system works as expected with valid inputs.
- **Negative Testing:** Verifies that the system handles invalid inputs and edge cases correctly without crashing.

Test Design and Execution:

Q11: How do you prioritize test cases in manual testing?

A11:

- **Critical functionality:** Test core features that are crucial to business processes.
- **High-risk areas:** Areas with frequent changes or high complexity.
- **User-focused features:** Features that are most used by end-users.
- **Time-consuming tests:** Automate or prioritize tests that are repetitive and take a lot of time manually.

Q12: How would you test a new feature in an application?

A12:

- **Understand requirements:** Review the documentation and discuss with the product team.
- **Create test cases:** Write test cases covering functional and non-functional aspects.
- **Test execution:** Execute test cases, including boundary and edge cases.
- **Bug reporting:** Log any issues and follow up for resolution.
- **Regression testing:** Ensure that new changes do not affect existing functionality.

Q13: How do you ensure that a test case is effective and efficient?

A13:

- **Clarity:** Make test cases easy to understand and execute.
- **Comprehensiveness:** Cover all possible input scenarios.
- **Reusability:** Write modular test cases that can be reused across different scenarios.
- **Conciseness:** Avoid unnecessary steps, ensuring test cases are brief and to the point.

Q14: How do you handle a situation where you don't have enough time to complete all tests?

A14:

- **Risk-based Testing:** Focus on high-priority, high-risk tests first.
- **Test Case Prioritization:** Prioritize based on functionality, frequency of use, and criticality.
- **Automation:** If possible, automate repetitive tests for faster execution.

- **Team Coordination:** Coordinate with developers to ensure critical functionality is tested first.

Q15: What are boundary value analysis and equivalence partitioning?

A15:

- **Boundary Value Analysis:** Focuses on testing the boundaries of input data (e.g., minimum, maximum, just below, and just above boundaries).
- **Equivalence Partitioning:** Divides input data into equivalent partitions where test cases can be derived from one representative value of the partition.

Q16: What are the different levels of testing?

A16:

- **Unit Testing:** Tests individual components or units.
- **Integration Testing:** Tests the interaction between components.
- **System Testing:** Tests the entire system as a whole.
- **Acceptance Testing:** Verifies if the system meets business requirements.

Q17: What is the purpose of regression testing?

A17:

Regression testing ensures that new changes (bug fixes, enhancements) do not negatively impact existing functionality of the software.

Q18: What is user acceptance testing (UAT)?

A18:

UAT is the final phase of testing where the end users test the software to ensure it meets their requirements before it is released to production.

Q19: Can you explain the difference between functional and non-functional testing?

A19:

- **Functional Testing:** Validates specific functionalities of the application based on requirements (e.g., login, search).

- **Non-functional Testing:** Validates non-functional aspects like performance, usability, security, and compatibility.

Q20: How would you test a web application's compatibility with different browsers?

A20:

- **Manual Testing:** Open the application on different browsers (Chrome, Firefox, IE, etc.), check for UI and functional issues, and report discrepancies.
- **Cross-browser testing tools:** Use tools like BrowserStack or Sauce Labs to perform cross-browser testing efficiently.

Bug Reporting and Communication

Q21: How do you report a bug, and what information do you include in a bug report?

A21:

A bug report should include:

- **Bug ID:** Unique identifier for the bug.
- **Summary:** A brief title or summary of the issue.
- **Description:** A detailed explanation of the issue.
- **Steps to Reproduce:** Exact steps to reproduce the issue.
- **Expected Result:** What should have happened.
- **Actual Result:** What actually happened.
- **Severity and Priority:** Severity (impact on the system) and priority (urgency of the fix).
- **Environment:** The environment in which the bug was found (e.g., browser version, OS).
- **Screenshots/Logs:** Attach any relevant screenshots or logs to assist developers.

Q22: How do you differentiate between a bug and a feature request?

A22:

- **Bug:** A bug is an unintended issue or deviation from the expected behavior that affects functionality.
- **Feature Request:** A feature request is a suggestion for improving the application, adding new functionality, or enhancing existing features.

Q23: What is the difference between severity and priority in bug reporting?

A23:

- **Severity:** Describes the impact or seriousness of the bug (e.g., critical, major, minor).
- **Priority:** Refers to how soon the bug should be fixed based on business needs (e.g., high, medium, low).

Q24: Can you explain how to reproduce a bug and why it's important?

A24:

Reproducing a bug involves following the exact steps to trigger the same issue again. It's important because it helps developers understand the issue clearly and verify its resolution.

Q25: How do you handle a situation where a developer does not agree with your bug report?

A25:

- **Discussion:** Engage in a discussion with the developer to understand their perspective.
- **Evidence:** Provide detailed evidence (logs, screenshots) to support the bug report.
- **Collaboration:** Work together to reproduce the issue and determine the root cause.
- **Escalation:** If needed, escalate to a senior or the project manager for clarification.

Testing Tools and Techniques

Q26: What is the difference between black-box and white-box testing?

A26:

- **Black-box Testing:** Focuses on testing the functionality of the software without knowledge of its internal structure. Testers interact with the software based on input-output behavior.
- **White-box Testing:** Involves testing the internal workings of an application, with knowledge of the code, logic, and structure.

Q27: How do you perform smoke testing and sanity testing?

A27:

- **Smoke Testing:** Quick, preliminary testing to ensure that the application is stable enough for more detailed testing. It checks if basic functionalities work (e.g., login, navigation).
- **Sanity Testing:** Focused testing to ensure that specific functionalities or fixes work after a minor change, often used after receiving a new build.

Q28: What is the difference between static and dynamic testing?

A28:

- **Static Testing:** Involves reviewing the code, documentation, or design without actually executing the program (e.g., code reviews, walkthroughs).
- **Dynamic Testing:** Involves executing the program to validate its behavior against expected outcomes.

Q29: What is a test environment, and how is it different from the production environment?

A29:

- **Test Environment:** A controlled environment set up specifically for testing purposes, where testers can validate the application without impacting the live environment.
- **Production Environment:** The live environment where the application is used by end-users, with real data and transactions.

Q30: How do you perform cross-browser testing manually?

A30:

To perform cross-browser testing:

- Open the application in different browsers (e.g., Chrome, Firefox, Safari, IE).
- Validate the UI, functionality, and performance on each browser.
- Report any issues related to rendering, layout, or features that don't work as expected.
- Use browser developer tools for debugging issues.

Advanced Manual Testing Questions

Q31: What is the purpose of a traceability matrix in testing?

A31:

A traceability matrix is used to ensure that all requirements are covered by test cases. It maps requirements to test cases, ensuring that each requirement is validated through testing.

Q32: How do you test a mobile application manually?

A32:

To test a mobile app manually:

- **Install the app** on different devices or emulators.
- **Verify functionality** (e.g., login, form submission, notifications).
- **Check UI/UX** for responsiveness and consistency across screen sizes.
- **Test performance** by launching the app under different conditions (network changes, battery use).
- **Verify compatibility** across multiple OS versions (iOS, Android).
- **Check security and permissions**, such as camera, location access, etc.

Q33: How do you test for security vulnerabilities in a web application manually?

A33:

- **Authentication Testing:** Test login functionality for vulnerabilities like weak passwords or session handling.
- **Authorization Testing:** Ensure that users cannot access resources they are not authorized to.
- **Input Validation:** Check for vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).
- **Data Encryption:** Verify that sensitive data is encrypted both in transit (HTTPS) and at rest.
- **Cookie Security:** Check if cookies are secured and have proper flags set (e.g., HttpOnly, Secure).

Q34: What is performance testing, and how would you test the performance of a system manually?

A34:

Performance testing evaluates how a system performs under expected load conditions. To manually test performance:

- **Load Testing:** Manually simulate normal user behavior to see how the system handles the expected load.
- **Stress Testing:** Test the system by pushing it beyond its limits to identify breaking points.
- **Usability Testing:** Observe the system's responsiveness and user experience under load.
- **Network Testing:** Check how the application behaves under different network conditions (e.g., slow networks, high latency).

Q35: How do you handle testing when there are multiple versions of the same application in production?

A35:

- **Version Tracking:** Maintain detailed records of the versions in production and ensure each one is tested independently.
- **Regression Testing:** Perform regression testing on the old and new versions to verify that updates do not affect existing functionality.
- **Compatibility Testing:** Ensure that new features work with previous versions if necessary.

Q36: What is risk-based testing? How do you perform it?

A36:

Risk-based testing involves prioritizing test cases based on the risk of failure and its impact on the system.

- Identify high-risk areas (critical functionality, frequently used features).
- Focus testing efforts on these high-risk areas.
- Use business impact and probability of failure to guide prioritization.

Q37: How would you test a payment gateway system manually?

A37:

- **Input Validation:** Test various valid and invalid payment details (credit card numbers, expiry dates).
- **Payment Processing:** Ensure that transactions are correctly processed and reflect in the system.
- **Error Handling:** Test invalid payment scenarios and verify appropriate error messages.
- **Security Testing:** Check that sensitive payment data is securely transmitted and stored.
- **Integration Testing:** Test the payment gateway integration with other systems (banking, third-party APIs).

Q38: What would be the testing approach for a multi-user system?

A38:

For a multi-user system:

- **Functional Testing:** Test user roles and permissions (admin, regular users, guests).
- **Concurrency Testing:** Simulate multiple users accessing the system simultaneously.
- **Session Management:** Verify that sessions are managed properly for each user.
- **Load Testing:** Ensure the system performs well with multiple users accessing the application simultaneously.

Q39: How do you ensure test coverage for all requirements?

A39:

- **Traceability Matrix:** Use a traceability matrix to ensure each requirement is mapped to test cases.
- **Test Case Design:** Write comprehensive test cases that cover both functional and non-functional aspects.
- **Review and Walkthroughs:** Collaborate with developers and stakeholders to ensure that all requirements are addressed.

Q40: How do you conduct a root cause analysis when you find a defect in production?

A40:

- **Reproduce the Issue:** Attempt to reproduce the defect in the same environment.
- **Log Analysis:** Check system logs for errors or patterns that could have caused the defect.
- **Collaborate with Developers:** Work closely with the development team to identify code or configuration issues.
- **Root Cause Identification:** Identify the root cause (e.g., coding error, configuration issue, or insufficient test coverage).
- **Implement Fixes:** Once the root cause is identified, fix the issue and perform regression testing to ensure that it doesn't happen again.

Requirement Analysis Scenarios

Q1: The requirements document is incomplete or unclear. How would you approach writing test cases or conducting testing in this situation?

A1: I would first communicate with the stakeholders (e.g., product owners, business analysts) to clarify the requirements. If clarification isn't possible, I would rely on exploratory testing based on similar features, the application's purpose, and past experiences. I'd document assumptions and get them reviewed to avoid misunderstandings.

Q2: A feature is being implemented but has no written documentation. How would you test it effectively?

A2: I would engage with the developers or product owners to understand the feature's purpose. Then, I'd perform exploratory testing to uncover potential issues and use tools like mind maps to structure my observations into test cases for future reference.

Bug Reporting and Resolution Scenarios

Q3: You report a bug, but the developer argues that it is not a bug. How would you handle this situation?

A3: I'd provide detailed steps to reproduce the bug, along with screenshots, logs, or video evidence if possible. If disagreement persists, I'd involve the product owner or relevant stakeholders to validate the issue based on requirements.

Q4: You find a critical bug an hour before the release. What actions will you take to handle this?

A4: I would immediately escalate the issue to the test lead and stakeholders, providing evidence and explaining the impact on users. Depending on the urgency, I'd suggest deferring the release or issuing a patch after release.

Test Case Design Scenarios

Q5: You are asked to test a login page. What are the key test scenarios and test cases you would cover?

A5: I'd test positive scenarios (valid credentials), negative scenarios (invalid credentials, empty fields), security aspects (SQL injection, brute force), usability (error messages), and performance (response time for multiple requests).

Q6: You need to test a payment gateway. What scenarios would you include in your testing?

A6: I'd include scenarios for successful payments, failed payments (e.g., insufficient funds, invalid card details), multiple payment modes, network failures during transactions, and proper error messaging.

Q7: A new feature is added to an existing application. How would you ensure that the new feature works without breaking existing functionality?

A7: I'd perform regression testing on impacted modules and critical workflows. I'd also test the new feature thoroughly with positive, negative, and edge cases to ensure it integrates seamlessly.

Regression and Retesting Scenarios

Q8: A bug you reported was fixed by the developer. How would you ensure that the fix is effective and hasn't introduced new issues?

A8: I'd retest the bug with the same steps and data. Then, I'd conduct regression testing in related areas to ensure the fix hasn't caused other issues.

Q9: A new build has been deployed with changes affecting multiple modules. How would you decide which areas to test first?

A9: I'd prioritize testing based on impact analysis, focusing on high-risk and frequently used areas first. I'd consult the development team to understand the scope of changes.

Agile Testing Scenarios

Q10: You are part of an Agile team, and requirements are continuously changing. How would you handle testing in such an environment?

A10: I'd stay involved in daily stand-ups to keep track of updates. I'd use a modular approach to test cases, enabling quick adjustments, and rely on exploratory testing to address dynamic changes effectively.

Q11: During a sprint, a high-priority bug is reported in production. How would you adjust your planned testing activities to address it?

A11: I'd pause lower-priority tasks and focus on reproducing and fixing the production bug. I'd also perform a root cause analysis to prevent similar issues in the future.

Real-World Application Testing Scenarios

Q12: You're testing an e-commerce website. A user places an order, but the payment fails. What scenarios would you test to troubleshoot this issue?

A12: I'd test the payment gateway integration, check server logs for errors, verify network connectivity during the transaction, and ensure appropriate error messages are displayed. I'd also test retry mechanisms.

Q13: You're testing a mobile app, and it works fine on one device but crashes on another. How would you investigate and report this issue?

A13: I'd check the app logs, device specifications, OS versions, and installed dependencies. I'd report the issue with detailed steps to reproduce and environment details.

Ad-Hoc and Exploratory Testing Scenarios

Q14: You're asked to test a new module, but you're given only a basic understanding of its functionality. How would you approach testing?

A14: I'd begin with exploratory testing to understand its behavior and identify potential issues. I'd create basic test cases based on observed functionality and validate them with stakeholders.

Q15: A stakeholder requests ad-hoc testing of a feature just before a demo. What approach would you take?

A15: I'd focus on high-priority areas, user-critical workflows, and visible functionality. I'd ensure the demo-critical paths are free of defects and communicate any risks identified.

User Experience and Usability Scenarios

Q16: You're testing a feature for accessibility compliance. What scenarios would you test?

A16: I'd test with screen readers, keyboard-only navigation, color contrast ratios, and alternative text for images. I'd validate compliance with accessibility standards like WCAG.

Q17: A user reports that they find the UI confusing for a specific workflow. How would you test for usability issues?

A17: I'd analyze the workflow for clarity, consistency, and navigation ease. I'd conduct user testing to gather feedback and identify areas of improvement.

System and Integration Testing Scenarios

Q18: Two modules developed by separate teams are integrated, but the system crashes when you test it. How would you debug and report this issue?

A18: I'd isolate the issue by testing each module individually, identify the integration points, and verify data exchange. I'd report findings with logs and steps to reproduce.

Q19: A third-party API integrated with your application returns intermittent errors. How would you test for stability and reliability?

A19: I'd simulate multiple API requests under different conditions, analyze response times and error rates, and check for proper handling of timeout scenarios.

Time-Critical Scenarios

Q20: You're asked to complete testing for a release within a very tight deadline. How would you prioritize your testing efforts?

A20: I'd focus on critical functionalities, high-risk areas, and user-facing workflows. I'd use exploratory testing to cover more ground quickly and document potential risks.

Test Environment Scenarios

Q21: The test environment is not stable, and you're unable to execute your test cases. How would you handle this situation?

A21: I'd document the issues, escalate them to the environment team, and work on areas that don't depend on the environment, such as reviewing test cases or preparing test data.

Critical Thinking Scenarios

Q22: You discover a bug that occurs only under rare conditions. How would you reproduce and report it?

A22: I'd analyze logs, trace steps leading to the bug, and vary inputs to replicate the issue. I'd provide detailed reproduction steps and the environmental setup in my report.

Q23: The development team fixes one bug, but it causes another. How would you address this recurring issue with the team?

A23: I'd suggest implementing better unit testing and code reviews. I'd also highlight the importance of regression testing and maintaining robust test cases for critical areas.